Real-time Simulation and Rendering of Large-scale Crowd Motion

---

A thesis
submitted in partial fulfilment
of the requirements for the Degree
of
Master of Science
in
Department of Computer Science
by
Bo Li

Supervisor: Dr. Ramakrishnan Mukundan
Co-supervisor: Dr. Tadao Takaoka

University of Canterbury
2013

Abstract

Crowd simulations are attracting increasing attention from both academia and the industry field and are implemented across a vast range of applications, from scientific demonstrations to video games and films. As such, the demand for greater realism in their aesthetics and the amount of agents involved is always growing. A successful crowd simulation must simulate large numbers of pedestrians' behaviours as realistically as possible in real-time. The thesis looks at two important aspects of crowd simulation and real-time animation.

First, this thesis introduces a new data structure called Extended Oriented Bounding Box (EOBB) and related methods for fast collision detection and obstacle avoidance in the simulation of crowd motion in virtual environments. The EOBB is extended to contain a region whose size is defined based on the instantaneous velocity vector, thus allowing a bounding volume representation of both geometry and motion. Such a representation is also found to be highly effective in motion planning using the location of vertices of bounding boxes in the immediate neighbourhood of the current crowd member.

Second, we present a detailed analysis of the effectiveness of spatial subdivision data structures, specifically for large-scale crowd simulation. For large-scale crowd simulation, computational time for collision detection is huge, and many studies use spatial partitioning data structure to reduce the computational time, depicting their strengths and weaknesses, but few compare multiple methods in an effort to present the best solution. This thesis attempts to address this by implementing and comparing four popular spatial partitioning data structures with the EOBB.

Table of Contents

Acknowledgments

First of all, I would like to thank Dr. Ramakrishnan Mukundan, for his precious advice, comments and insightful discussions. Thanks to his guidance during my research and his patience to revise and review my writing many times without his patient and insightful guidance, continuous support and wide knowledge, this thesis would have been impossible. It is my greatest pleasure to be the research student of Dr R. Mukundan.

I would like to thank my wife and parents for supporting me throughout my education and for their unique form of encouragement during difficult times.

Last, but far from least, I appreciate the help from the University of Canterbury throughout the development of my Masters project. Thanks for providing me a stable and safe environment for my study.

Chapter I

Introduction

In general, a crowd is "a large group of individuals in the same physical environment sharing a common goal who may act in a different way rather than when they are alone" [59]. According to the paper of Ulicney and Thalmann [62], crowds have become an important research area for many scientists since the nineteenth century. In particular, from the perspective of computer graphics research, crowd simulation refers to an artificial creation of virtual crowds that mimic the behaviour of real crowds, which is an active area of research that finds several applications in the design of urban environments and the development of emergency evacuation strategies [53]. Recent research in this field has focused primarily on crowd dynamics and behaviour models [15], [24], spatial subdivision methods [51], perception models [63] and path/motion planning algorithms [73]. More specifically, our simulation focuses on the motion generation of individual characters, and running in real-time with large-scaled pedestrians. The behaviour of a character, in this context, is then defined as change of motion choice that the character makes over time. Therefore, crowd simulation is defined as the collection of every individual behaviours over time.

1.1 Motivation

This section explores the motivation underlying our research by enumerating important crowd simulation applications.

Because we can see crowds anywhere at any time, crowd simulation is vital in many virtual environment applications that concern education, training, and entertainment. For example, a movie might use a special effect to create a battle scene where many characters are fighting each other. A virtual environment, such as a city for training fire fighters, will look more convincing if there exists a large crowd populating the environment, just as in real

cities. In architecture, crowding is also an important factor for designing a building or for urban planning. Computer games, war-strategy games in particular, require many characters who interact with each other and exhibit high-level behaviours, such as grouping or separating. In light of these specific applications and the impressive demand for graphics content, there is every reason to believe that demand for simulated crowds will grow in the future.

Although crowd simulation can be applied to a wide range of applications, there are several challenges that make high-quality crowd animation a difficult task to carry out. Firstly, most motion planning techniques today offer solutions too expensive to simulate very large crowds. To keep high frame rates, such methods either require to lower the number of characters, or to reduce their rendering to 2D points. We want to design a motion planning solution in a complete crowd engine, so that the simulation can run in real time. Secondly, it is not easy to maintain the real-time performance of a large simulated crowd. These restrictions come from the limited computational resources. Our goal is to be able to simulate crowds with large sizes in real-time.

1.2 Objectives

The objectives of this research are:

- To be able to detect and avoid collision fast and accurate. The detection and the avoidance of both character collision and collision between characters and obstacles should be efficient. In particular, the inter-character collision testing speed should be faster than simple frame-by-frame method. Given initial and target constraints, such as a pose, position, orientation and time duration, an algorithm must synthesize the motions that meet those constraints in real time or without any noticeable time delay.

- To develop a platform to evaluated the proposed algorithms. The platform should have a flexible architecture to accommodate different acceleration data structures for the purpose of testing the research ideas.

It should also be possible to create simulation scenarios for different places and situations easily.

- To find the best acceleration data structure so that a large-scaled crowd simulation can run in real-time. Different data spatial data structures need to be implemented, and measure the success and effectiveness of each spatial partitioning data structure. The simulation methods need to be implemented in a comparable fashion, and in a similar environment. The focus will be simulating a large number of pedestrians in real time, with environment and pedestrian representation as secondary considerations, to allow maximum processing to be devoted to the acceleration method.

1.3 Thesis Overview

The rest of this thesis is organized as follows:

Chapter 2 first provides an overview of what components make a crowd simulation, and explores the related work. Then we thoroughly review previous work related to acceleration data structures.

Chapter 3 discusses the considerations for how the scene will be displayed and presented.

Chapter 4 presents the construction of the extended bounding volume and its application in collision detection and speed adjustment with further explores the usefulness of the proposed structure in obstacle avoidance and local direction position.

Chapter 5 describes space partitioning data structures considered in our research: Grid, Quadtree, k-d tree, and Bounding Interval Hierarchy (BIH). Following by the implementation aspects of the above four data structures.

Chapter 6 presents an experimental evaluation of results using flow fields in motion planning. The chapter then lists the results of the comparative

analysis between four acceleration data structures.

Chapter 7 summaries the findings of this research. It concludes the thesis and outlines some future directions in the area.

1.4 Thesis Contributions

The main contributions of this thesis are

- Design of a new data structure called Extended Oriented Bounding Box for geometry and motion representation.
- Development of efficient collision avoidance and motion planning algorithms for scenes involving obstacles.
- Development of a complete framework for large-scale crowd rendering in several constrained environments
- Analysis of several spatial partitioning structures proving the efficiency of the regular grid structure for large-scale crowd rendering

1.4.1 Publications

1. The following paper has accepted for oral presentation and published in the conference proceedings:
    - Ramakrishnan. Mukundan, Bo Li. "Crowd simulation: extended oriented bounding boxes for geometry and motion representation" in Proceedings of the 27*th Conference on Image and Vision Computing New Zealand*, 2012, Pages 121-125.
2. The following paper has been submitted to the 30*th* Conference on Computer Graphics International:
    - Bo Li, Ramakrishnan. Mukundan. "Comparative Analysis of Spatial Partitioning Methods for Large-Scale, Real-time Crowd Simulation"

Chapter II

Background

Crowd simulation is a large topic that covers many areas. As a result, many different sub-problems are part of crowd simulation. Each of these sub-problems is an active research field on its own. A review of research related to crowd simulation will be presented in this chapter. In section 2.1, we will give an overview of general crowd simulation techniques. The latter parts of the chapter more formally explore the main subject of this thesis, looking deeply into collision avoidance and the acceleration algorithms behind them. An overview of general collision detection algorithms is given in section 2.2. In section 2.3, we will recap the popular spatial partitioning data structures.

2.1 Crowd Modelling

Crowds are ubiquitous in real life. From the perspective of computer graphics research, crowd simulation refers to the simulation of the behaviour of real crowds using 2- or 3D artificial creations of virtual characters. Our simulation focuses on simulating large numbers of individual motion characters in real-time, and each character can change his motion behaviour over time. In this section, we will review existing work for each important aspect of crowd simulation.

In crowd simulation, the words, agent, character, and pedestrian all refer to a single member of a crowd.

2.1.1 Crowd Rendering

Although our research does not focus on the visual rendering of crowds, crowd rendering is important in crowd simulation applications, from urban planning to entertainment. However, rendering thousands of characters and virtual scenes in real-time is still challenging. The main reason is that rendering is the most time consuming job in crowd simulation. Several methods have been

used to reduce the complexity of crowd rendering. This section overviews the current character rendering techniques.

Luebke et al. proposed that the appropriate model or resolution is selected usually based on the objects distance to the camera in 2002 [12] Brogan and Hodgins use simulation level-of-detail (LOD) to control the movement and actions of large groups [11]. OSullivan et al. proposed the idea of conversational and social behavior LOD [45]. In 2004, Ulicny et al. used OpenGL display list and OpenGL Scene Graph 3D toolkit to optimize the crowd rendering [5]. In the same year, De Heras Ciechomski et al. improved on their performance by using 4 level-of-detail meshes for their model [7]. Millan and Rudomin presented a GPU-based crowd rendering platform in 2007 [17]. They can render 100000 characters at interactive frame rate.

We will not discuss crowd rendering in detail in this thesis, as this is not necessarily part of our focus. However, we implemented both 2D and 3D characters into our simulation applications, and simple cubes are used to model a character to minimise the computational time. The description of implementation is in Chapter 3.

2.1.2 Path Finding

When routing an agent through an environment we are charged with two different tasks. The first is finding a global route from the start position of the agent to the desired goal position.

If we had a map of the environment, this would be relatively easy. The book of Lavalle gives an excellent overview on this topic [56]. A roadmap can create a realistic path planning graph by incorporating knowledge of the environment. A roadmap is a graph that connects to each point in the environment through a simple path. Besides this, the graph should also consist of only one connected component. If the graph adheres to these properties, we can connect any point A in the environment with any other point B through this graph. From point A we can find a simple path to the graph. By travelling over this graph we can get to a point C, which will have a simple path to the goal position B. Thus we obtain a global path for the agent.

Many different techniques exist to create this graph. Usually the A* algorithm [71] or Dijkstra's algorithm [41] is used to compute the required path through this graph. In this thesis we will not dive deeper into this subject as it will not affect our main topic. Our model is able to facilitate these extensions. In future work these extensions might be included.

2.1.3 Behaviour Modelling

Pedestrians having proper behaviour is very important for crowd simulation applications. Without behaviour, the pedestrian does nothing but stand still. For example, when a person tries to leave a building, he or she will find the shortest path first, then avoid colliding with all obstacles when walking to the exit. In crowd simulation, behaviours like collision avoidance and motion planning are referred to as pedestrians' behaviours. In recent years, the personal computer has become more powerful, and it is possible to simulate crowds with thousands of pedestrians where each one has more specific behaviours. Depending on the level of detail of the simulation model, pedestrian behaviour modelling can be classified into three different types: microscopic, mesoscopic and macroscopic models [47].

Crowd simulation in a microscopic model is the most detailed. Each pedestrian pursues its own behaviour in the most realistic way possible. The pedestrian is determined by an exact position and velocity. Collision avoidance is very important. Every pedestrian in this model should avoid each other. It is not acceptable to have the pedestrian walk straight through the obstacles. These models enable users to investigate interactions of individuals in a crowd to the building details, emergencies, crowd density, and actions of other pedestrians.

In the macroscopic model, the crowd is represented as a fluid with specific densities and average velocity. Pedestrians in this model no longer exist: the model considers the flow of the crowd, instead of individual behaviour for each pedestrian, and the user is only interested in distribution of densities for the simulation. Macroscopic models use the relation of density to walking speed and flow to calculate overall movements of the crowd. The macroscopic model is used in [28] to reduce traffic congestion, and in [23] to observe the

flow of autonomous agents along a passage without modelling or tracking individual agents. This method is useful for example in the study of traffic conditions, but is inaccurate for pedestrian simulation.

In the mesoscopic model, pedestrians are represented individually with their own goals, though we are not interested in the exact behaviour of each pedestrian, but instead the overriding characteristics, such as velocity distributions. We can be sure that the pedestrian is likely to be somewhere in the vicinity of their position. Collision avoidance is less relevant. The pedestrian behaviour is simulated by altering its speed depending on the local density. This model can be used for measuring throughput of agents and density build up [54].

We are mainly interested in the microscopic model in this thesis; we want to have realistic behaviour for each pedestrian, and pedestrians should avoid all collisions. Microscopic behaviour modelling has three main approaches:

Social Forces Model

The social forces model represents the microscopic behaviour of a person in a crowd by the social fields. The first mathematical model is proposed in [14]. In this model, the motions of moving pedestrians are described as if they are subject to external social forces. These forces are a measure for internal motivations (collision avoidance, etc.) of the individuals. This model successfully simulates the self-organization phenomenon of crowd and pedestrian behaviours.

The motion of an agent a with the mass m_a in the social forces model is composed of the sum of several social force terms $f_a(t)$ and $p_a(t)$, denoting fluctuations which correspond to individual variations. Taken together, these terms either result in the acceleration or deceleration of each agent. In interplay with other agents or the environment, their effect is either repulsive or attractive.

$$m_a \frac{dv_a}{d} = f_a(t) + \xi_a(t) \tag{2.1}$$

The social forces model has been used to simulate many important crowd phenomena successfully, such as improved prediction accuracy

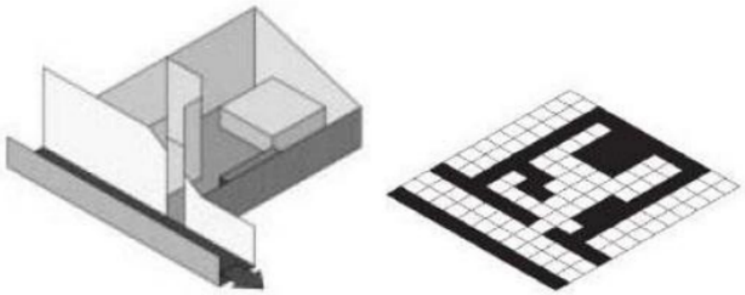

Figure 2.1: A structure represented by a cellular automaton model, with black squares signifying impassable grids. Adapted from [44].

and analysis of interactions between pedestrians [30]. However, it does not always produce realistic results because the model considers people behave based on physical laws when collisions happen. Human do not completely follow the laws of physics; they behave based on their thoughts and cognitive processes of their minds.

Cellular Automate Model

A cellular automaton (CA) is a discrete model using a collection of cells in grid architecture, where each cell has a finite number of states, k, that it can be in, in a finite number of dimensions, according to a set of rules [68]. The number of states k is typically an integer with $k = 2$ (binary). Every cell has the same rule for updating, based on the values in this neighbourhood. Each time the rules are applied to the whole grid, a new generation is created. The cellular automata method can be used to simulate pedestrian flow. It is fast and relatively simple, with the walkway represented by a cellular grid. Fig. 2.1 shows a building represented in a cellular automaton model.

a CA was used in [27] which looked at pedestrian movement within a shopping centre. In this model the movement was directed only toward the destination and could only change at decision points with pedestrian interactions not being considered.

Due to the resolution of cells within the grid, the densities allowed

within each area of a given space are usually less than that which would be allowed in real equivalents. These models discretise the underlying simulation space as a flow grid. Meta-data which can be used for decision making or routing for the agents is associated with each cell on the flow grid. Graph search algorithms may be used to search for paths towards goal points on the flow grid. The CA approach does not involve agent interactions: flow on a particular grid cell is only available if the target cell it leads to is not occupied by other agents. Li-jun et al.[37] present good examples of approaches using cellular automata, a 2.5 dimensional CA models is proposed, and they proved their CA model is practicable and reasonable for crowd evacuating simulation

CA however is discrete and restricts the movement of the agents to the next available cell only. Simulation of the agent's speed is hence difficult and discrete.

Rule-Based Model

Rule based models [9] introduced specific behaviour based rules for simulating the movements and interactions of simple creatures like birds, fishes and animals in the form of a flock. This model was later used to simulate movements of pedestrians. In this model creatures interact based on their perception of the environment. These behavioural rules include collision avoidance, velocity matching and flock centring. Most rule-based approaches do not model physical interactions due to collisions but rather they apply simple wait rules to avoid collisions all together. Rule-based models can be very realistic for low density crowds but they are not physically accurate. It is also much easier to use a rule-based approach in combination with cognitive models. Terzopoulos and Shao [67] proposed such an extended rule-based model, which includes motor, perceptual, cognitive and behavioural aspects of the individuals altogether. In [66], a vision model was proposed, the different rules added to agent based on the surround environment. Wen et al.[70] also proposed a rule-based behaviour model, a set of simple navigation rules are defined for each agent. The authors' results shows that the crowd can move on the large scale space in real-time.

2.1.4 Motion Planning

Motion planning is another important aspect in making pedestrian behaviour more realistic. The goal of motion planning is to generate the desired trajectory to be fed into the motion control system so that the pedestrian executes or tracks the desired trajectory. In this section, we give a brief overview of prior work on motion planning. The popular methods for motion planning include reactive approach, trajectory parametrisations and exhaustive search. The reactive approach was pioneered in [32] and its basic idea is to assign potential fields to obstacles and pathways to expel the trajectory away from obstacles and bring the trajectory to the final destination. Follow-up work can be found in [13]. For parametrisation approaches, a general principle is outlined in [6]. In [49], trajectories are parametrised by polynomials, then coefficients are determined by fitting the kinematic model and boundary conditions. For exhaustive search methods in [4], the space is typically divided into regions and a safe path is found by successively searching adjacent regions to the destination. To avoid the need for complete prior knowledge of the environment, an improvement has been proposed in [33], which combines exhaustive search with the potential field methods for 3D path planning in a dynamic environment.

2.2 Collision Detection and Avoidance

Collision detection detects if two or more objects are intersecting. More specifically, collision detection concerns the problems of determining if, when, and where two objects come into contact [1]. It is fundamental to many varied applications, including computer games, physically based simulations (such as computer animation), robotics, virtual prototyping, and engineering simulations. For the purpose of our research, the collision detection problem will be limited to its applicability in crowd simulation applications only.

Collision detection is very important to get immersive performances in crowd simulation, which can help pedestrians avoid intersecting with other obstacles. In a real-time crowd simulation application, there are always lots of moving pedestrians, and this dynamic feature makes the collision detection problem hard. Rapid collision detection algorithms are required to realize

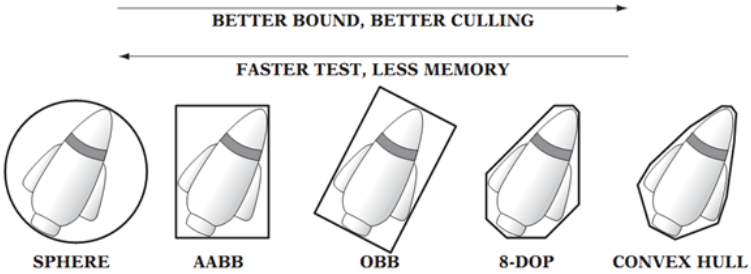

Figure 2.2: Types of bounding volumes: sphere, axis-aligned bounding box (AABB), oriented bounding box (OBB), eight-direction discrete orientation polytope (8-DOP), and convex hull. [1]

real-time performance.

In a crowd simulation, where pedestrians need to change walking direction or stop moving before colliding with others, collision detection is performed to determine whether two pedestrians will interfere with others. The exact contact point between them is not important for crowd simulation, so testing the complex geometry of two pedestrians for collision against each other is not necessary and bounding volumes have widely been used to avoid expensive testing. Simple geometrical objects such as spheres and boxes are used to represent pedestrians of more complex natures. The bounding volume (BV) includes spheres [26], axis-aligned bounding boxes (AABBs) [72], oriented bounding boxes (OBBs) [25], and k-DOPs [74]. AABBs and spheres are simple primitives which provide smaller computational costs and faster overlap tests. OBBs and k-DOPs can yield tighter fits, but are relatively expensive for overlap tests. Figure 2.2 shows that there are trade-offs involved between BV complexity of the topology and performance. In practice, AABBs are most commonly used. They can be handled very efficiently without any numerical stability issues during construction and in most cases enclose primitives reasonably well. OBBs and cylindrical bounding volumes are generally found suitable for character models in highly dynamic scenes [10], [57].

2.3 Acceleration Methods

Spatial partitioning techniques are widely used in both ray tracing and collision detection areas. It divides space into regions and tests if objects overlap the same region of space, so the number of pair comparisons is reduced drastically. Several approaches have been proposed to using space partitioning data structure in crowd simulation to avoid the $O(n^2)$ complexity of the proximity queries between entities.

2.3.1 Uniform Grid

Uniform grids are a simple and efficient data structure which divide space into cells and keys are assigned to the respective cell. [36] first applied grids to 3D range queries. Recent research on uniform grids considers the memory requirements and parallelization techniques. [34] proposed a compact representation of uniform grids and discussed its application in GPU-based ray tracing. In [31], efficient parallel grid construction is considered in the context of ray tracing. They proposed an algorithm for which performance does not depend on the primitive distribution, because the construction problem is reduced to sorting pairs of primitives and cell indices. Turk [60] uses a uniform grid approach to determine collisions between molecular models represented by collections of spheres. He states that a uniform grid approach is perfectly suitable in this special case, since the radii of organic atoms do not differ significantly and therefore a small upper bound for the number of atoms per grid cell exists.

The hash table can be used to reduce storage requirements. Overmars [46] employs a uniform grid approach combined with hashing to accelerate point location in fast subdivisions of n-dimensional space (subdivisions that contain no long and skinny cells). He derives general upper bounds both on storage requirement and point location time. Mirtich [39] employs a hierarchical hash table to efficiently detect intersections of axis-aligned bounding boxes which are used to represent swept volumes of moving objects over a time interval. His approach requires a pre-processing step to determine a range of appropriate hash table resolutions. Teschner et al. [58] proposed an optimized spatial hashing technique for the collision detection of deformable

objects. They use a uniform grid approach to accelerate collision detection for deformable models consisting of tetrahedra and are the first to give a detailed analysis of all involved parameters, including hash table size and optimal grid cell size.

2.3.2 Quadtree

Quadtree is an indexing data structure that subdivides each region of space into four equal-size sub-regions. This subdivision is recursively performed for each sub-region until a desired number of triangles are achieved within each region. A quadtree has been constructed and used in [55] which accelerates the performance of collision avoidance. Quadtree has been popular for use in paths planning, with [19] using a qudatree for paths planning of robots, where it can speed up the selection of better paths, and find relevant paths precisely. [29] describes how the quadtree approach can be used to store multiple sensors data to model a large driving area. They build the quadtree bottom-way up so that it is easy to implement geographical operations such as neighbour finding, searching and updating. In [66], a hybrid quadtree waypoint approach is used for macro level path finding, and the bounding box is used to represent the entire scene. The scene is then recursively divided into four sections, stopping only when the maximum number of levels is reached or when the content of leaf nodes is entirely homogenous, containing exclusively accessible or inaccessible regions.

Finding the neighbours of a specific leaf node is fundamental to the operation for tree data structures and it is very important to have fast and accurate neighbour search methods in crowd simulation to reduce unnecessary computations for collision quires. In quadtrees, finding neighbours takes O(r) computational time for the worst case [50]. An algorithm for labelling connected components based on quadtrees is also proposed in [36]. The algorithm takes almost O (N) computational time for the worst case, where N is the number of leaves in the quadtree. Recently, [30] proposed a new graph-based data structure which represents the quadtree for a given binary image. It takes less execution time than a normal quadtree, but the limitation of it is that each node of the quadtree can only have at most eight edges.

The uniform nature of quadtrees allow them to be implemented in a very compact manner. However this comes at the expense of having less flexibility in choosing how the data is sub-divided because of the fixed nature of the quadtree subdivision.

2.3.3 K-D tree

K-D trees are an indexing data structure that sub-divide a region of space into two sub-regions at a set distance along an axis. Like quadtrees, this can be recursively done for each region; however, the distance and axis of sub-division is changed on a region-by-region basis in order to maintain approximately the same number of triangles within the sub-regions. The k-d tree, initially introduced by Bentley [3], aimed to generalize a binary search tree to high-dimensional data. It has been widely applied to search in a low-dimensional space to accelerate ray tracing and collision detection, and has been observed to give superior performance in ray tracing compared to other acceleration structures [21], especially when built using the surface area heuristic (SAH) [38].

Many researchers are looking for ways to build k-d trees efficiently. Havran [22] presented an Automatic Termination Criteria to determine when to stop the recursive building and a split clipping method to get better candidate split plane, thus improving the quality of the k-d tree (increased by 1%-35%), but a large number of intersections reduced the speed of construction. Shevtsov [52] presented a minimum-maximum binning algorithm which first split the scenes, then detected the split plane to get the approximate minimal cost of k-d tree. The problem with this approach is that it is difficult to get the accurate number on both sides of split plane, and as a result, the approximate minimal cost will be far from the optimal one. Finally, the k-d tree can also be used in path finding to improve the search efficiency. [69] uses a k-d tree structure in the implementation of the A* algorithm to reduce memory usage and speed up the search.

The k-d tree is a well-utilized nearest neighbour search algorithm. Its average computational complexity of searching the Nearest Neighbour (NN) is O (log n) in the case of randomly distributed model points, where n is the

number of nodes in the k-d tree. The worst computational complexity is O $(kn^{l-1/k})$ [35].

2.3.4 Bounding Interval Hierarchy

Bounding Interval Hierarchy (BIH) [8] have been recently introduced and used for real-time ray tracing. And it was first presented under the name of SKD-Trees[43], and BoxTrees[20] independently invented by Zachmann.

Bounding interval hierarchies (BIH) exhibit many of the properties of both bounding volume hierarchies (BVH) and k-d trees [8] . Whereas the construction and storage of BIH is comparable to that of BVH, the traversal of BIH resembles that of k-d trees. Furthermore, BIH are also binary trees just like k-d trees. Finally, BIH are axis-aligned as are its ancestors. [65] proposed a way to just use the BIH data structure for the construction phase but traverse the tree in a way a traditional axis aligned bounding box hierarchy does. This enables some simple speed up optimizations for large ray bundles while keeping memory/cache usage low.

To construct any space partitioning structure some form of heuristic is commonly used. [8] described a simplistic "global" heuristic, which only requires an axis-aligned bounding box, rather than the full set of primitives, making it much more suitable for a fast construction.

Chapter III

Crowd Simulation and Animation

In this chapter, the general considerations of crowd simulation are introduced in Section 3.1 and include rendering, camera, lighting and visualization. In Section 3.2, the graphical implementation of pedestrians is presented. Lastly, the different scene models are described in Section 3.3.

3.1 General Considerations

3.1.1 Rendering

To build a large-scaled crowd simulation in real-time, fast rendering is important. We use a single-faced polygon to represent the wall to reduce the complexity of the models, then the codes for the walls are added to a Display List. This allows for all of the scene's vertex and pixel data to be cached during program initialisation and placed into a compiled state in memory. Each time a frame is redrawn to the screen, OpenGL can call the Display List for the buildings without needing to recalculate lighting's normal. This allows far more computational power to be freed for drawing the agents, which are the main focus of the scene. The only disadvantage in the use of Display Lists is that the geometrical data within the list cannot be manipulated during run time, having been compiled during the initialisation of the application. The buildings are part of a fixed scene and this can be assumed as a low risk issue. Also the buildings are neither animated nor changed for the duration of the simulation.

Texture is another aspect which can affect the rendering time, but since it is not necessary in the context of the aims of this research, we do not use any texture techniques.

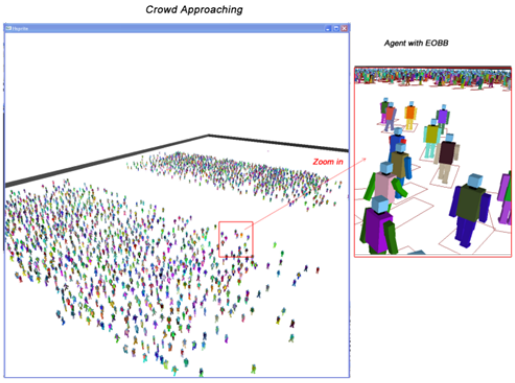

Figure 3.1: Screenshot of a typical 3D scene. It is possible to zoom in and display the agents' trajectories, which is useful for analysing the behaviour of the agents.

3.1.2 Camera

To have a good analysis of the crowd during simulation running, the camera has to be controlled such that we have the various viewpoints and angles. The position of the viewpoint is used as the camera's centre axis, allowing the viewer to navigate the simulation at any angle. The movements of the scene are controlled by mouse and keyboard. After the camera is set up, we can use the keyboard to control our viewpoint, zoom in and out to check the micro and macro model of the simulation, and we can also rotate the camera to see the simulation from different angles. Quaternion camera system [64], which can easily translate mouse movements into camera rotations by analysing the movement of a mouse cursor within the window, is implemented. An example is shown in Fig. 3.1, the left-side screenshot shows the macro model of view, and the right-side shows the detail of pedestrians.

3.1.3 Lighting

Lighting is one of the most expensive operations in 3D computer graphics. At the same time, lighting makes the 3D model more realistic. To best achieve high speed rendering and a realistic scene, flat shading is used. This helps the viewer understand a level of scene depth, making the scene seem less

two dimensional when viewed. A level of low ambient light was applied to all objects, bringing their material brightness up to an acceptable minimum level. A single light source was then added to the location of the viewpoint, meaning that as we navigate through the scene, the light source would move along with the view. This would ensure that proper lighting is present from every viewpoint, eliminating the risks of unlit areas and poor visibility. With this scheme in place, we do not have to implement multiple lights in the scene.

3.1.4 Visualization

Since we are going to design a crowd model which can be used in macro and micro level simulation, the 2D crowd does not present enough behaviour details. Therefore, the 3D model is used in our simulation and the implementations are presented in next section. To get better analysis results of the crowd's motion, the flow field and density map are also implemented. Fig. 3.2 depicts the flow field of one experiment: two crowds moving in opposite directions. Each line records an agent's motion during simulation, so the motion of agents can be seen clearly in the flow field, such as change in direction or arrival of the target. The density map is another way to show the distribution of densities. The different colours are used to represent density: high density areas have a brighter colour, and a darker colour means lower density (Fig. 3.3).

3.1.5 Components of the Simulation

The simulation model consists of the following components:

- Agent (or Pedestrian): A single member of a crowd, which has a set of properties, such as velocity, goal, direction, neighbours, etc.

- Obstacle: Consists of a few walls. The agent in the vicinity of the obstacle must avoid colliding with it by finding a new path based on the location of the wall

- Goal (or Target, Destination): The location where the agents want to go.

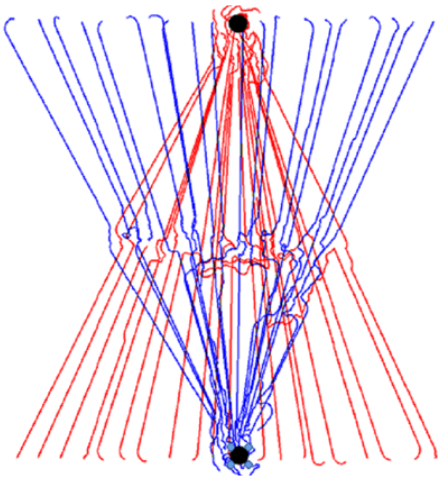

Figure 3.2: Flow field showing crowd motion towards targets in a scene

Pedestrian movement is built up by motion planning algorithms (Chapter 4.1)

- Motion planning: Between all agents there are three extended oriented bounding boxes (EOBBs) that update the velocity when the distance is decreasing. Detail will described in Chapter 4.

- Obstacle Avoidance: All pedestrians in the vicinity of the obstacle will change the direction with the offset angle when the distance is decreasing. (Disscussed in Chapter 4.2)

The basic concept of simulation is described here. For each agent in the simulation, we calculated the velocity of the agent first and use the velocity to get its new position, finally moving the agent to this new position. This is simple, but generic, which makes it possible to attach modules for all kind of behaviours and simulation features.

3.2 pedestrians

The most essential part of the scene will be the agents themselves. In our research, the focus is on the flow of agents and the performance of the large-scale simulations in real-time. Therefore agent density is more important

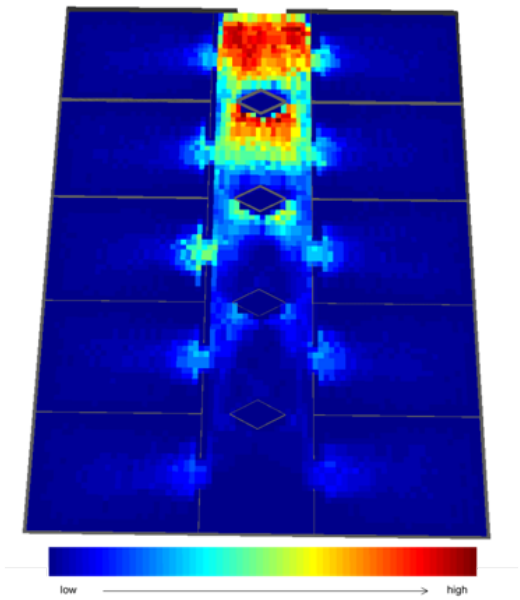

Figure 3.3: Density map. Red colour represents high density areas, blue colour represents low density areas.

than graphical quality, so the pedestrians' graphical representation needs to be as simple as it can be made.

Initially, to build our simulation model, we used a sphere to represent pedestrians, an inbuilt primitive from the GLUT library. The pre-compile is normal and without errors, and can be invoked with a single line of code. The diameter of the sphere was set to be equal to the intended human model size, and in proportion to the implemented structures. With low graphical overhead, the inbuilt sphere proved to be a perfect three dimensional representation for designing and adjusting the behaviour. Fig. 3.4 shows a screenshot from an early design stage.

To make the simulation more realistic, human models are used. A complex 3D model (such as mesh model) is not used as they are too complicated and out of the purview of the aims of our research. Articulated representation is used to depict a 3D pedestrian model. Fig. 3.5 shows the various viewpoints of a single 3D pedestrian. Articulated objects consist of rigid parts which are linked by joints to form an object skeleton. The movements

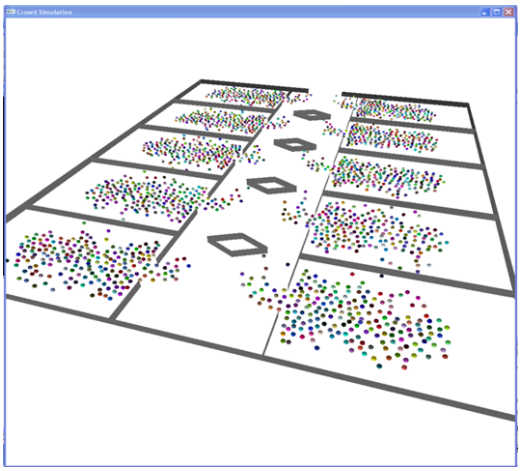

Figure 3.4: A basic building scene hosting 2000 agents, moving along paths and running in real-time.

of these articulated objects and models are restricted and do not have the freedom as might be the case with human body parts and movements. The model's positions can be changed in real time and only require altering the values of a small set of parameters to get the desired animation. To achieve easy and fast implementation, we use a hierarchical tree model [2] to represent pedestrians. The torso of a robot is used as the root of the tree, shown in Fig. 3.6. In the hierarchical model, the edges contain geometric transformations and the nodes contain geometry and possibly drawing attributes. To draw the pedestrians, we used a pre-order traversal method, so we start with the left branch, follow it to the left as deep as we can go, then go back up to the first right branch, and proceed recursively. To program the scheme, we use stacks to store the required matrices and attributes as we move through the tree.

To make pedestrian move, a key-frame animation is adopted. The animator positions the pedestrian at a set of times (the key frames). In hand animation, animators fill in the remaining frames, a process called in-betweening. In computer graphics, we can automate in-betweening by interpolating the joint angles between the key frames or, equivalently, using simple approximations to obtain the required dynamic equations between

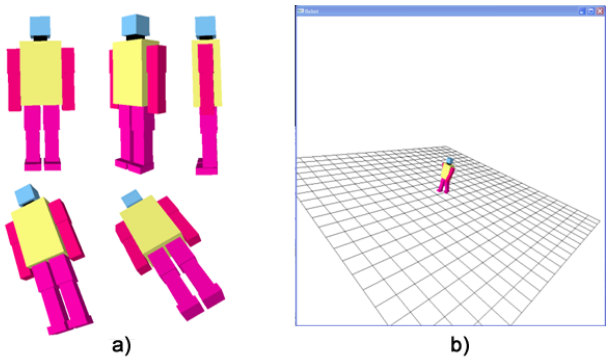

Figure 3.5: A 3D pedestrian. a) The different views of pedestrian are displayed. b) A pedestrian walking on the ground.

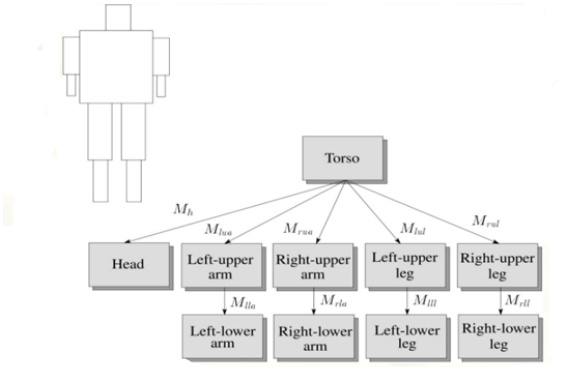

Figure 3.6: A humanoid figure and the tree representation. An agent consists of cubes of different size. Each cube is connected in tree structure.

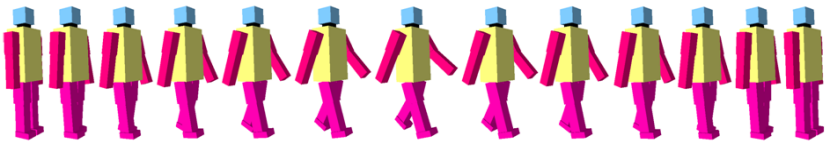

Figure 3.7: Synthetic walking corresponding to the sequence of a pedestrian.

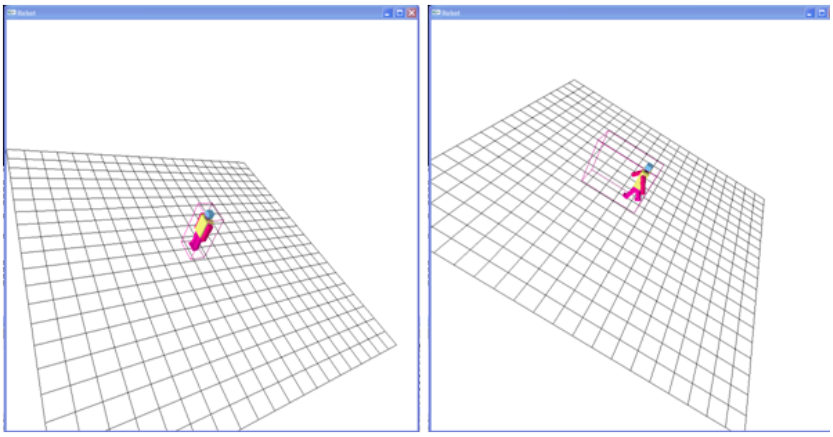

Figure 3.8: Walk-cycles and the corresponding stride lengths. a) Pedestrian stands without moving b) A Pedestrian walking briskly with longer stride length.

key frames. Fig. 3.7 depicts a key frame.

Two walk-cycles are assigned to each pedestrian, Fig. 3.8 shows the situation of the pedestrian standing and walking on the scene. Fig. 3.9 show the screenshots of different viewpoints of both slow and brisk walking of the pedestrianss. The detail of the walk-cycles implementation are described in the next chapter.

3.3 Scene Models

To better facilitate research into the different aspects of behaviour modelling and acceleration data structures, the virtual environments need to allow for manipulation to show large-scale movements of pedestrians so that proper behaviour can be fit into it easily. The simulation model should be able

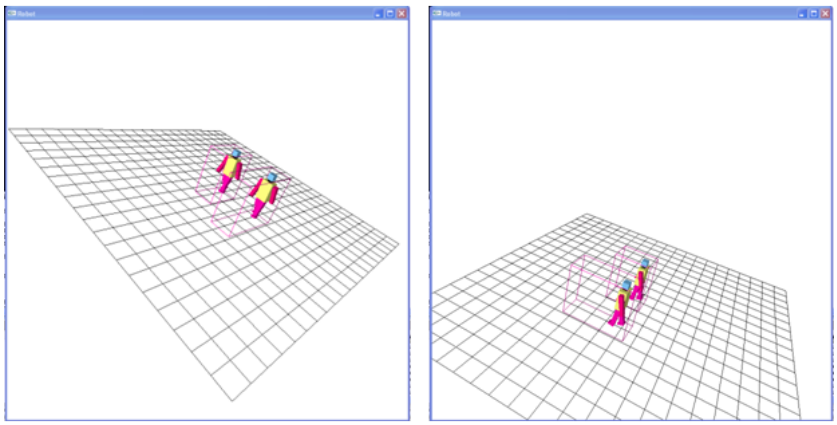

Figure 3.9: Different viewpoints of pedestrians with two walk-cycles

Environment	Name
a corridor with and without obstacles	E1
a highly constrained environment	E2
a large hall with one obstacle	E3
a theatre with ten lecture rooms	E4
a large-scaled open space	E5

Table 3.1: List of environment names

to offer scope for a variety of obstacles and sizes, with the option to add areas of open space. This gives models the desired behaviour and allows for acceleration data structures to be implemented with sufficient diversity for a fair test. The scene needs to be large enough to accommodate a fairly dense population, such as 5,000 pedestrians, so that we can test performance for each acceleration data structure. For these reasons, five environments were chosen for the testing and demonstrational purposes of our research. Table 3.1 lists the name of the environments.

Fig. 3.10 shows the top-down view of the first environment (E1) used to examine the behaviour model. It is a simple corridor and two crowds are able to move in opposite directions. Obstacles can also be added to the corridor to check the obstacle avoidance algorithm.

The second environment is named E2 (Fig. 3.11) and is a highly con-

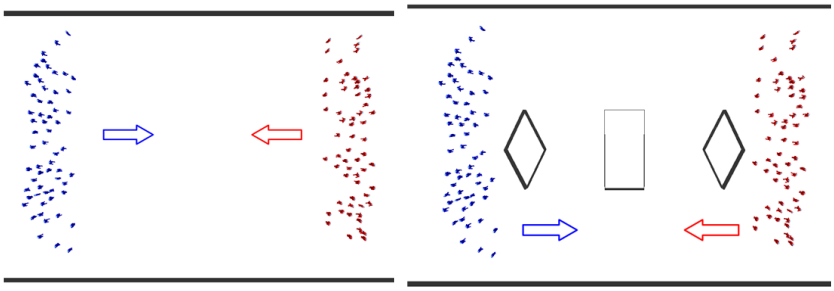

Figure 3.10: E1: corridor with two crowds, and different obstacles can be added

Figure 3.11: E2: a highly constrained environment: the wall with various direction and an obstacle in the middle limited the size of the path way

strained environment where a number of obstacles are defined to limit the movement to a small region. This scene contains the congestion area with great complexity and can be used to demonstrate excellent performance of our path finding and obstacle avoidance algorithm.

The third scene E3 is shown in Fig. 3.12: a real-life environment, a large hall with one exit, is represented. A simple obstacle is added inside the hall. It can be used to test our new path finding algorithm. The simulation results show that all pedestrians in the hall are able to navigate freely and easily find the free path to walk out of the hall.

The fourth scene is also a real-life environment and combines E1 and E3

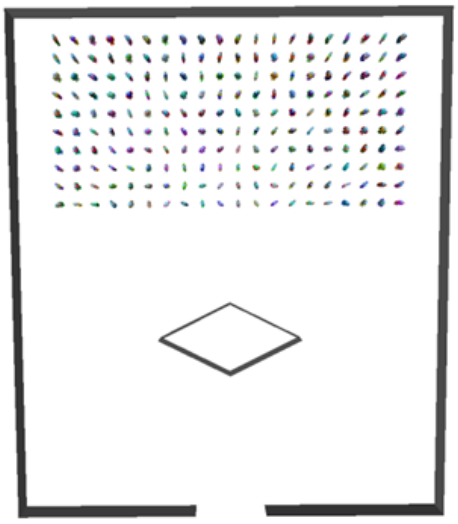

Figure 3.12: E3: 200 pedestrians located in a hall.

to model a lecture theatre. The large theatre includes ten lecture rooms and each room can accommodate various numbers of students. The theatre has a long corridor and each lecture room is connected to this corridor. Each side of the corridor has five lecture rooms. Finally, the theatre has one exit at the end of the corridor. with this model, both motion planning algorithm and acceleration data structure can be evaluated in the same time. The top-down view of this environment is shown in Fig. 3.13.

Finally, we build a large open space which can contain large-scale pedestrians (Fig. 3.14) so the performance of different acceleration data structures can be checked and tested.

These five environments provide a range of small and large open spaces with various obstacles of different sizes and shapes. The complexity of the scenes should allow for plenty of opportunity to see a full range of behaviours displayed from the collision avoidance and implementations of acceleration data structures.

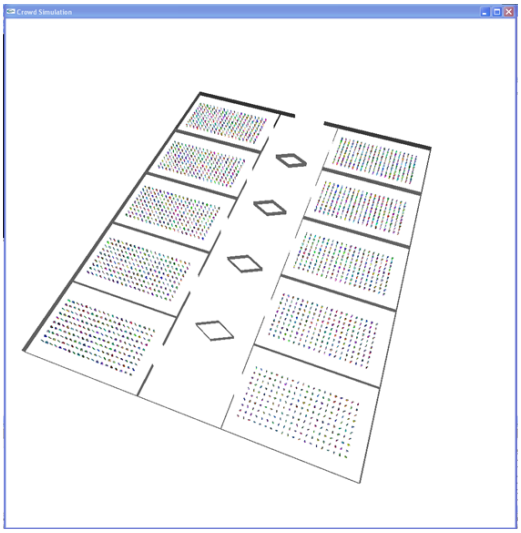

Figure 3.13: E4: a large theatre, including ten lecture rooms

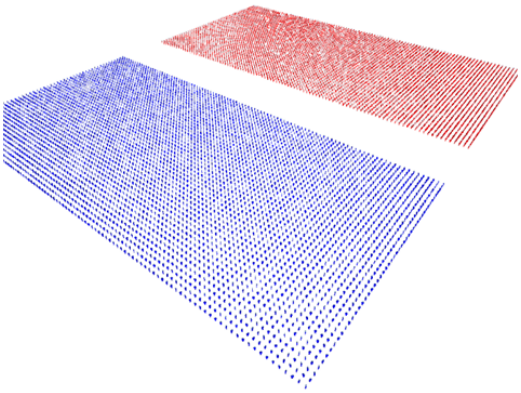

Figure 3.14: E5: large-scaled scene. 10,000 pedestrians are in the scene

Chapter IV

Motion Planning Using Bounding Volumes

In this chapter, we present an efficient motion planning method for characters to avoid collision and ways to find the free path. This method uses a novel extended oriented bounding box (EOBB) to represent a pedestrian with motion and rule-based motion planning to find an optimised path for speed and direction. The chapter is split into three sections. Section 1 presents the EOBB in collision and for speed adjustment. The rule based motion planning is covered in Section 2. Finally, Section 3 further explores the usefulness of the EOBB in obstacle avoidance and local direction control based on the location of vertices surrounding the current obstacle.

4.1 Extended Oriented Bounding Volume(EOBB)

When the simulation includes many pedestrians that are animated simultaneously, efficiency is an important concern. Collision detection between articulated pedestrians is computationally intensive and may not be necessary for most of the cases due to the aims of the crowd simulation. Therefore, the bounding box is used to simplify a pedestrian geometry in collision detection. Oriented bounding box (OBB) and Axis-aligned bounding box (AABB) are the two most common bounding box methods used. An AABB can be represented with just minimum and maximum extents along each axis, whereas an OBB representation must encode not only position and widths, but also orientation. The advantage OBBs have over AABBs as bounding volumes is that they can bind their enclosed geometry more tightly. In particular, OBB is best for character models in highly dynamic scenes. Therefore, we used an OBB to enclose the pedestrian.

The most common solution for collision detection is discretely sampling the pedestrians' locations along the motion path and performing local collision detection for other locations [26]. Due to their discrete nature, these al-

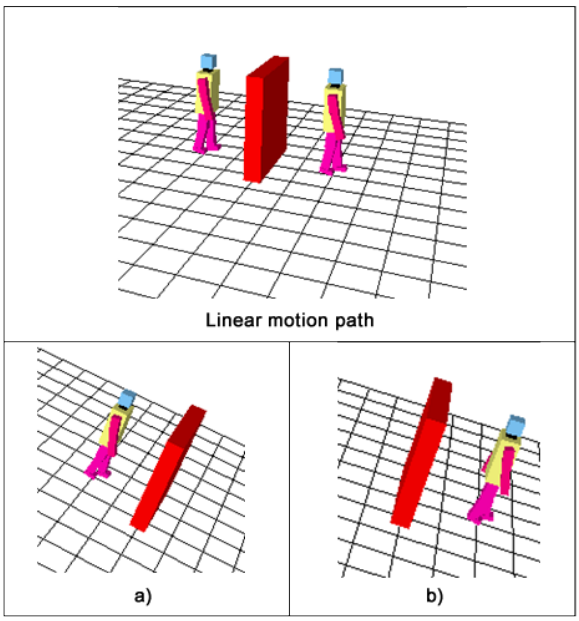

Figure 4.1: Linear motion collisions. (a) The time of entry collision between the pedestrian and wall, and (b) the time of exit collision between the pedestrian and wall, if they were to continue their initial motion, as if no physical collision occurred.

gorithms perform poorly with small pedestrians or fast motion (Fig. 4.1). To overcome the limitations of discrete collision detection methods, we propose a simplified and flexible data structure called extended oriented bounding box (EOBB), which can be easily constructed based on the current position and velocity dependent stride length and direction of a pedestrian. In [57], Sung proposed a similar spatio-temporal volume called the motion oriented bounding box (MOBB) and used a hierarchical structure for collision detection. This method is specifically suited for crowd animation driven by motion capture data since the nodes of a MOBB tree store both the space and time bounds of a motion clip. In our research, instead of a bounding volume hierarchy, we use four spatial partitioning data structures for broad-phase collision detection to find the best acceleration data structure, the detail of which will be discussed in Chapter 5.

The motion of a crowd is often represented by velocity fields, where both direction and speed of a member at any instant can vary depending on the

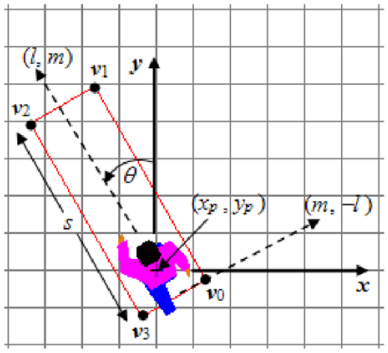

Figure 4.2: Definition of the extended OBB. The agent's geometry and motion are bounded by the oriented bounding box

position of obstacles in the area, the target location, and the behaviour model used. This information can be attached to the OBB of the member by including the space defined by one stride length at the current speed in the current direction. The definition of the EOBB is shown in Fig. 4.2. The orientation of the box is defined by the angle from the y-axis. The box is aligned to two orthonormal directions (l, m) and (m, -l) on the xy-plane, where $l = -\sin\theta$, and $m = \cos\theta$. The vertices v_i ($i = 0..3$) of the OBB can be easily obtained using the current position (x_p, y_p) of the character model and stride length s. We assume that the motion is confined to the xy-plane, and therefore only the two-dimensional projection of the OBB is considered.

The stride length depends on the instantaneous speed of movement and the number of frames used in a walk-cycle. Two walk-cycles representing "slow" and "brisk" walk and the corresponding stride lengths, s_1 and s_2 are shown in Fig. 4.3. In an animation sequence, both the stride length and direction are constantly recomputed based on the results of overlap tests and the number of vertices in the immediate neighbourhood of each character (more details in Section 3.2).

Once the EOBB is defined, we will introduce a simple and fast method of intersection testing. Intersection testing of two EOBB is very similar to testing standard OBB trees. Note that we are seeking only yes/no intersection queries, and hence can exit as soon as an intersection is found. We

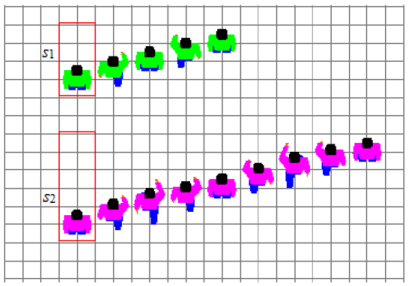

Figure 4.3: Two walk-cycles and the corresponding stride lengths.

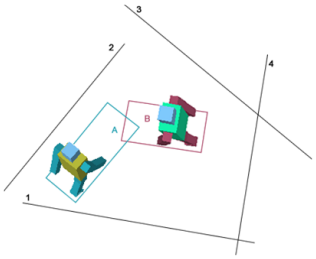

Figure 4.4: Two agents are bounding by EOBB, four separating axis are parallel to one of the EOBB's edges

use the separating axis theorem [40] for the two-dimensional case to perform the overlap test between the two EOBBs. It states that two OBBs do not intersect if and only if there exists a separating axis such that the projections of the two polygons onto the separating axis do not intersect. Fig. 4.4 illustrates two OBBs, A and B. There are only four axes tests required. Four separating axes are parallel to one of the OBBs edges. For example, suppose we want to check if OBB A intersects with OBB B. We first project A and B onto the *axis 1*, and if the projected intervals do not overlap (Fig. 4.5), there exists a separating axis, so the OBB A do not intersect with OBB B. Otherwise, we need to continue checking *axes 2*, *3* and *4*. Finally, if the projected intervals overlap in all four directions, then the boxes are considered as intersecting (Fig. 4.6).

The result of the overlap test is used to progressively refine the stride lengths of the intersecting EOBBs, thereby adjusting the speeds of the two

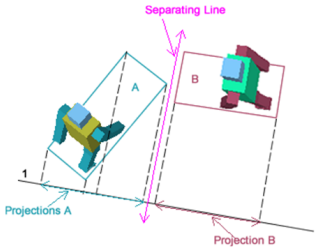

Figure 4.5: Polygons A and B do not intersect because there exists a separating axis such that the projections of A and B onto the axis do not intersect.

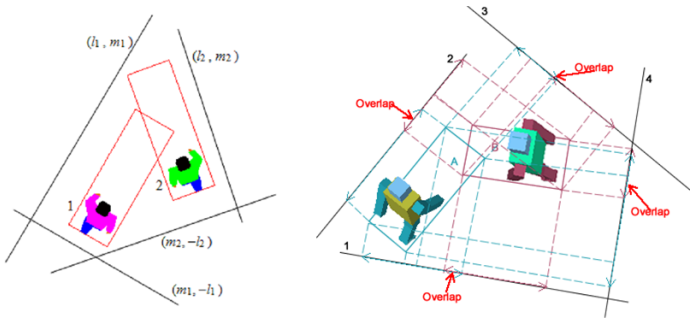

Figure 4.6: Overlap test for the EOBBs on 2D and 3D models, both cases the two agents collide

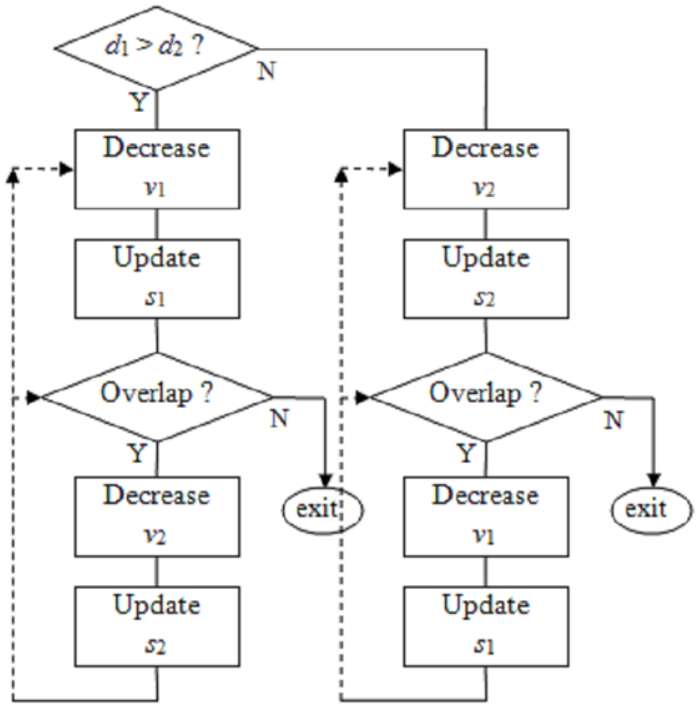

Figure 4.7: Iterative update of speed and stride length using the results of the OBB overlap tests.

characters. In this algorithm, we assign a higher priority to the character closer to the target, allowing it to move first or at a faster speed. This approach is particularly useful in situations where a dense crowd moves towards a single target. With reference to Fig. 4.6, if d_1 and d_2 denote respectively the distances of characters *1* and *2* from the target position, with v_1, v_2 the current discredited speeds, then the speeds are reduced in steps according to the algorithm given in Fig. 4.7, to determine if there exists EOBBs of reduced lengths that do not overlap.

If such a pair of EOBBs exist, then the corresponding stride lengths are used for the next step. Otherwise the speed (and the stride length) of both characters in the current direction are set to 0.

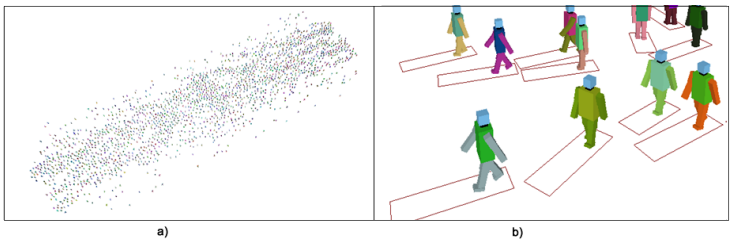

Figure 4.8: Macroscopic and microscopic of crowd simulation. a) the flow of the crowd is only consideration . b) The simulation detail can be seen clearly, and the behaviour for each agent is able to be observed.

4.2 Motion Planning

The motion planning needs to be implemented such that the pedestrians move around the scene from a location to a target without colliding with others. The pedestrians are represented as individuals in this approach and act for themselves and interact with other pedestrians and obstacles based on behavioural rules. For example, if a pedestrian is walking close to a wall, he needs to find a new direction before moving, otherwise he will stop moving until there is a free path.

The microscopic model was developed so that we can investigate crowd's individual behaviours as well as the motion flow. For example, if we wanted to model a large-scaled crowd, we could have gone for the fluid dynamics approach (Fig. 4.8a), but at the same time we wanted to be able to zoom in on a small area and follow individual pedestrians and see what they specifically were up to and to examine their individual behaviours (Fig. 4.8b).

We want to the motion planning can be simple implemented, at the same time, can detect collision and avoidance it faster and accurate. Based on the literature review, the rule-based model is simple to be developed, which also could present accurate and realistic movement of crowds. Therefore, we proposed a rule-based motion planning for agent-based crowd simulation, where a pedestrian quickly chooses velocity so as to achieve the desired goal as well as avoiding oncoming collisions accurately.

The motion planning method in our model is used for both obstacle avoidance and local direction control. Here we consider three possible directions of

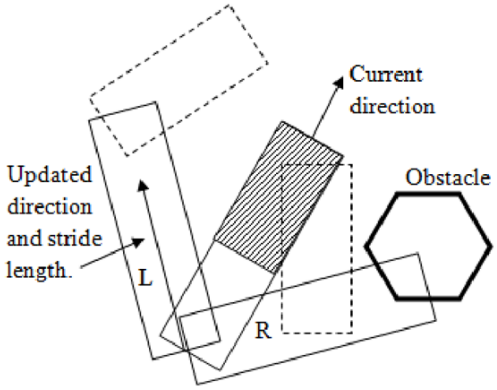

Figure 4.9: The process of updating the current direction and speed uses three EOBBs and the locations of points inside the regions.

motion at a given location (x_p, y_p): the current direction (l, m) and two other directions given by a 45 degree turn towards each side. These directions and the corresponding OBBs are indicated by L, R in Fig. 4.9. The stride length in the current direction is decided by the output of the algorithm in Fig. 4.7. This is indicated by the shaded region in Fig. 4.9. The stride length in the other two directions is set to the maximum possible value (as shown in Fig. 4.2). The extended OBBs of other characters in the neighbourhood (dotted lines in Fig. 4.9) are compared with the three bounding volumes to determine if any of them intersect with any of the three regions. If they do, then the distance of the point of intersection closest to the current position (x_p, y_p) is recorded. The same process is repeated for the edges of any obstacles in the vicinity of the character. Based on the minimum distance of points of intersection in each region, the direction of motion and stride length for the next iteration are computed.

It may be noted that when a course deviation as described above is effected, it is commonly followed by a stepwise course correction, forcing the character to start moving towards the intended direction defined by the target location. Taking this aspect into consideration, if two directions are found to be equally probable, the one which results in minimum deviation from the target is chosen.

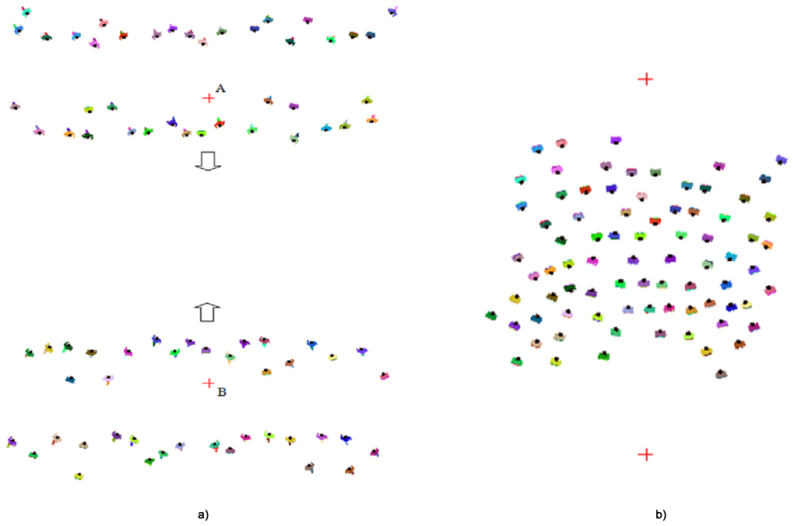

Figure 4.10: A deadlock condition of two crowds moving in opposite direction. a).Two crowds moving in opposite direction. b)Deadlock situation when direction updates are not used

A similar rule-based motion planning algorithm can be found in [73]. Our method differs from that given by Xiong [73] in that it uses only the extended OBBs and considers only three possible directions. The three directions are found to be sufficient to resolve deadlocked conditions as the one shown in Fig. 4.10, where two crowds move in opposite directions towards targets A and B. With the help of the above algorithm, the characters on the periphery of the crowd will find new directions of motion, giving more space for other inner members of the crowd.

Obstacles can be conveniently represented by a set of vertically oriented quadrilateral elements (or walls) with the two end points on the ground plane and the height as parameters. On a two dimensional plane, they are represented simply by a collection of segments of straight lines. Each line segment has a unique angle ω defined exactly like the direction angle θ (see Fig. 4.2) of a character element.

A collision avoidance scheme can be designed using the model of a "repulsive force" for each wall implemented as a course deviation from the current direction θ when a character approaches a wall. The modified value of θ is

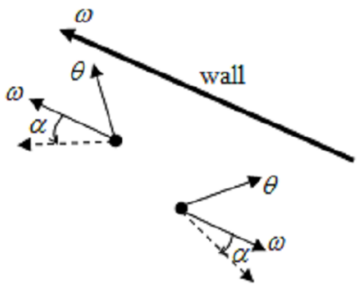

Figure 4.11: The direction of movement of a character is updated as it approaches a wall.

computed as given in Eq. 4.1 below. The offset angle α is chosen as inversely proportional to the distance of the current position P from the wall. Two possible cases are shown in Fig. 4.11.

$$\begin{aligned} \theta &\leftarrow \omega + \alpha, \quad if\, 0 \leq \omega - \theta \leq (\pi/2), \\ \theta &\leftarrow \omega + \pi - \alpha, \quad if\, (\pi/2) \leq \omega - \theta \leq \pi, \end{aligned} \tag{4.1}$$

Chapter V

Acceleration Algorithms

Acceleration algorithms involving spatial partitioning methods are extensively used in crowd simulation for real-time collision avoidance. This chapter presents a detailed analysis and evaluate the effectiveness of different implementations of spatial subdivision data structures, specifically for large-scale crowd simulation.

5.1 Introduction

One of the key problems for collision detection with a large crowd model is that they are time-consuming, and this makes it almost impossible to run a simulation on real-time. The use of distance checking helps to alleviate this issue by avoiding complicated checks for most distant obstacles; however, this scheme can go only so far. Collisions always happen between the obstacles around the agent and often only a small subsection of the obstacles have the chance to collide. This means much of collision detection may not be necessary. In order to avoid these issues, a data structure is needed, one that helps limit the amount of collision detection while at the same time having a fast search time. This is where the use of a spatial hierarchy comes into play. Spatial hierarchy is a way of organizing data such that two objects that are near to one another in space remain near one another in the data structure. Having such a data structure greatly aids in quick performance of the aforementioned culling tasks because large swaths of the hierarchy can quickly be culled and the detailed processing can be limited to only those areas where it is truly needed. A fast culling algorithm is nothing without an efficient and compact data structure for storing the spatial association information. In the crowd simulation, the positions of pedestrians change

frequently, and it is therefore necessary to focus on both querying and updating performance of the spatial partitioning data structure at the same time. There are many different types of spatial hierarchies, and four different acceleration data structures were discussed with their own strengths and weaknesses in the following section, which included grid, k-d tree, quadtree, and bounding interval hierarchy (BIH). The first three data structures have been widely used in broad-phase collision detection and the fourth method [8] is used primarily in ray tracing applications.

5.2 Grid

In this section, a uniform grid is introduced first and the spatial hashing is given in order to provide better memory management.

5.2.1 Uniform Grid

The basic idea of uniform grids is relatively simple: the simulation space is uniformly subdivided into cubic cells and the objects are, based on their bounding volumes, assigned to these cells. Since all grid cells are cubes, all their edges are of equal length. This edge length is referred to as the size of the cell. The Axis-Aligned Bounding Box (AABB) is used for each object, so the size of an object is referring to the length of the longest edge of its corresponding AABB.

Two different strategies exist for inserting objects into grids:

1. Object is assigned to all the cells which the bounding volume of an object overlaps (Fig. 5.1b)

2. Object is assigned to one single cell only (Fig. 5.1a).

The first strategy has a major drawback: because the object is assigned to all of the cells which it overlaps with, the collisions need to be detected iterating through all the cells that the object occupied. Therefore, the same collision between two particular objects might be detected multiple times. In contrast, since objects can only be assigned to one single cell, finding all occurring collisions is reduced to only checking each object against the objects that are stored in the same or any of the directly adjacent cells.

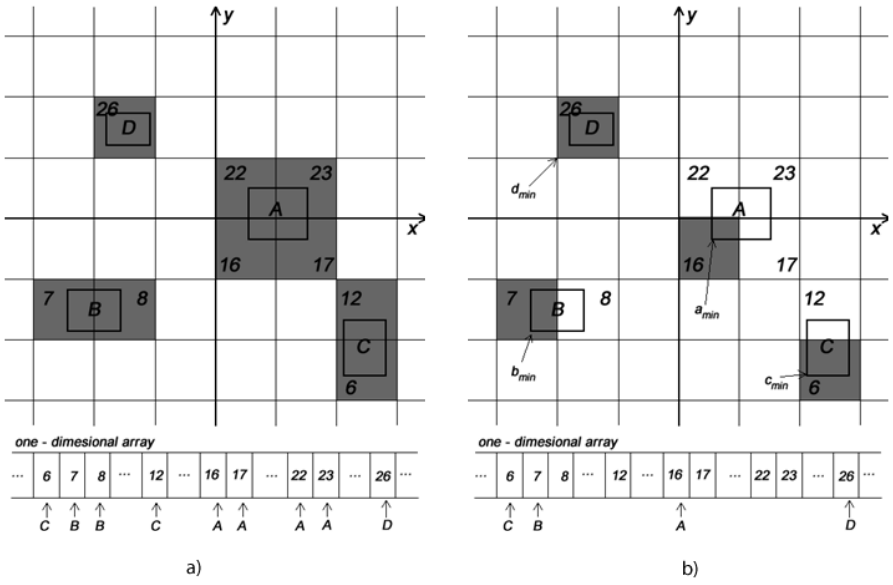

Figure 5.1: Two strategies of inserting objects into grids. a) Object is assigned to all the cells which the bounding volume of an object overlaps. b) Object is assigned to one single cell

In order to achieve the second strategy, two important aspects need to be considered. Firstly, the size of the grid cells must be larger than the largest object of the simulation (Fig. 5.2). Secondly, the point of reference used to calculate the cell association must be identical for every object. For instance, the centre of the AABB or either of both vertices can be used, but they must never be mixed. For example, in Fig. 5.2a, the lower corner of the AABB is used in order to assign objects to cells. In Fig. 5.2b, using the lower corner as point of reference leads to object A being assigned to cell number 2 and object B being assigned to cell number 1. If the upper corner is used, object A is assigned to cell number 4 and object B is assigned to cell number 3. In either case, both cells are adjacent and thus the intersection is detected. If, however, object A would be assigned to cell number 4 based on the upper corner of its AABB and object B would be assigned to cell number 1 based on the lower corner of its AABB, the intersection of both objects could not be detected if only adjacent cells are tested.

After the method of insert object is selected, the grid is implemented

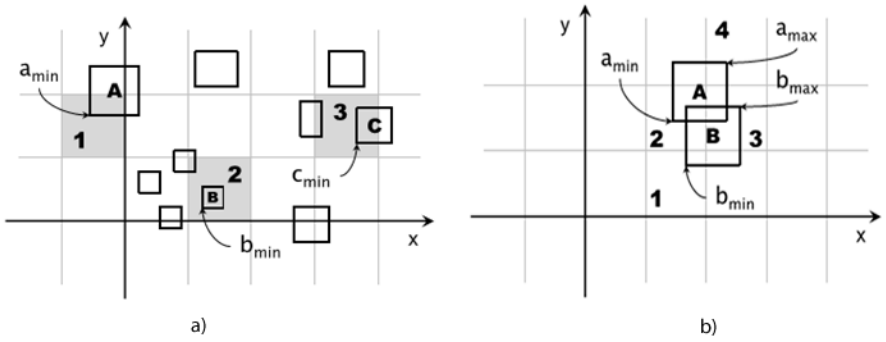

Figure 5.2: Issues related to assigning objects to one single cell a) Using the lower corner of the AABB: object A is assigned to cell number 1, object B to cell number 2, and object C to cell number 3. b)In order to associate objects with particular cells, the same point of reference must be used for every AABB.

based on a linear array of cells. Indexing follows the intuitive pattern that is illustrated in Fig. 5.3. Each cell has to store references to all the objects that overlap the cell. Usually, the references are stored in either linked lists or dynamic arrays.

5.2.2 Hashed Storage

One issue that has not been addressed in uniform grid is the hash table in large-scaled crowd simulation, when there are many objects scattered in the large scene. The grid requires many cells to contain the objects, which will undoubtedly exhaust main memory resources. For example, if the memory requirement of each single grid cell was a few bytes and if 10,000 cells were required in each dimension, this would already sum 10^8 cells in a two-dimensional simulation, and thus the data structure for this grid alone would claim memory of almost one gigabyte. Hashed storage is used to solve this problem (Fig. 5.4). A hash function maps the three-dimensional (or two-dimensional) cells of the infinite grid to a one-dimensional hash-table of finite size. For example, a point with index of position $p = (i, j, k)$ is hashed into a hash table of size m by computing its cell index c as follows:

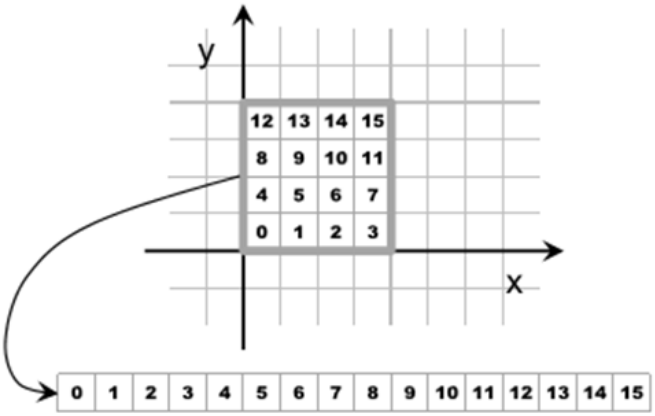

Figure 5.3: A two-dimensional uniform grid represented by a one-dimensional linear array. Indexing in combination with a two-dimensional hash grid follows the exact same pattern.

$$c = ((i \times u) \oplus (j \times v) \oplus (k \times w)) \bmod m \tag{5.1}$$

where u, v , and w are large prime numbers. If multiple points are hashed to the same hash cell, chaining is employed to resolve these hash collisions; that is, the points are stored in a linked list specific to this cell. An example is shown in Fig. 5.4.

5.2.3 Updating the Data Structure

Presently, most of the necessary aspects concerning the conceptual design of a hash grid have already been discussed. In this section, we will discuss how to update the grid data structure. Since the objects in crowd simulation are moving around all the time, the hash table needs to be updated in every time step. The two approaches that realized such an updated strategy are as follows:

1. All objects are simultaneously removed from the data structure and immediately afterwards they are reinserted based on their current locations, which may or may not have changed since the last time step.

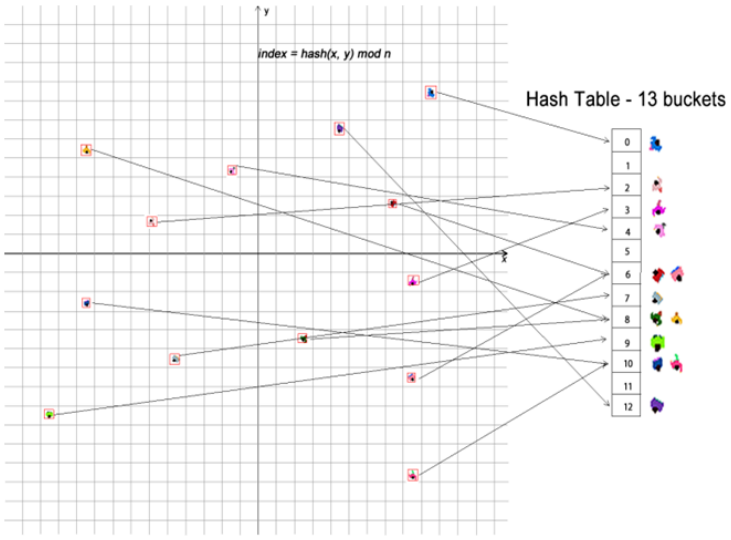

Figure 5.4: Example of mobile objects with a grid-based partitioning, and a hash table storing objects' indices

It is very easy to implement. Removing all objects is equivalent to clearing all grid cells, which can be done efficiently as long as one keeps track of all the object-occupied cells while inserting objects. Moreover, reinserting an object is no different from initially inserting an object; hence, the exact same methods may be used.

2. For every object, the hash value is recomputed based on the object's current spatial location. If the new hash value is identical to the hash value of the previous time step, the object remains assigned to its current grid cell. If the hash value changes, the cell association has to be changed too, meaning the object has to be removed from its currently assigned cell and stored in the cell that corresponds to the new hash value.

Consequently, when using hash grids the collision detection is always separated into two different stages: the updated phase followed by the detection step. In every time step, the data structure is first updated as described above in order to adapt to the current object distribution. Afterwards, all the occurring collisions are detected using up-to-date data structures.

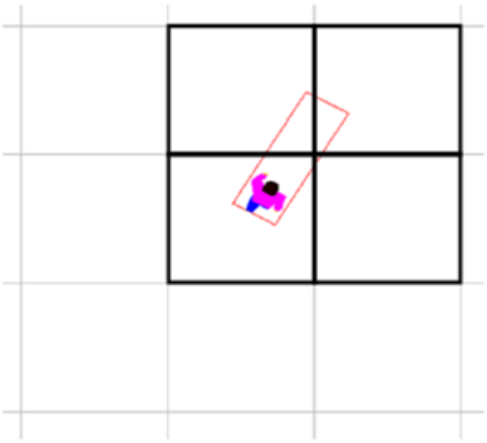

Figure 5.5: Spatial partitioning using a regular grid, where grid size equals maximum stride length.

5.2.4 Implementation

In this section, we discuss the implementation details and the influence of parameters on the efficiency of the approach.

The cell size influences the number object pairs that have to be tested for intersection. The traditional methods require that cell size to be larger or equal to the average size of all objects, so each object can only be stored in one cell. In our implementation, the grid size is defined as equivalent to a predefined maximum stride length (Fig. 5.5), which has been used in our EOBB (chapter 4) data structure.

In the next step, we need to select a hash function, which should have fast computational speed and avoid hash collisions. [16] indicated that simple and fast-to-execute hash functions are generally preferable for spatial hashing and result in the fastest possible collision detection time. Based on their research, we used an XOR hash function (*e.q.* 5.1) to generate the hash table, with the values of 73,856,093 for u and 19,349,663 for v.

In designing the hash table, we need to consider the trade-off between number of cells and memory usage. If we reduce the number of cells, the probability of objects being assigned to the same cell of the hash grid increases. Based on the analysis in [16], our hash table has a size equal to

the number of the objects in the crowd simulation, and therefore, collision detection time can get very stable and small.

After we define all the parameters of spatial hashing, we can insert and delete the object from the grid. During the simulation, each object in the simulation is moving around at each time step, so we need to update the data structure at each time step. Because the object has limited moving distance for each time step (usually less than the size of a cell), most of objects will stay in the same cell for a few time steps, so instead of rebuilding the data structure each time, we only need to update the grid where the object has moved to a different cell. Therefore, the second updating approach is the strategy of choice for updating the data structure.

After the hash spatial data structure is created, the grid neighbour searching method is used for finding all potential colliding agents. The position (x_p, y_p) of a character can be directly used to compute the hash table index as well as the indices of the neighbouring grids. The direction (l, m) is used to select a maximum of three neighbouring cells out of a total of 8 (Fig. 5.5). If the maximum and minimum world coordinate of scene spaces are given by (x_{min}, y_{min}), (x_{max}, y_{max}), and if the maximum stride length used for discretization is s, then the index k of the cell containing the character with position (x_p, y_p) is given by:

$$k = \left[\frac{x_p - x_{min}}{s}\right] + M\left[\frac{y_p - y_{min}}{s}\right] \tag{5.2}$$

The neighbouring cells are selected based on the signs of the components of the current direction vector (l, m). For example, the cell vertically above index k given by the index k + M is chosen if $m \geq 0$. The cell diagonally above the current cell is given by k + M + *1* and is selected if both l and m are positive. Only four identified cells (including the current cell) are used for the broad-phase collision detection for each agent.

Algorithm 1 details every step of the collision avoidance process by using grid data structure. The first step is to find pedestrians that can potentially collide. We take advantage of the grid structure covering the whole environment: at runtime, every pedestrian is registered in its current grid cell. To keep the algorithm fast, the steps mentioned above are implemented: the

middle point of pedestrians is used, and the pedestrians only register in one cell. Then four directly adjacent cells are searched to find the neighbours.

Algorithm 1: Grid Implementation

```
Input: Agents, Grids G, Maxmim allowed distacne between two
       object d
begin
    foreach Agent a in Agents do
        register a into its current cell G[a]
    end
    foreach Agent a in cell G[a] do
        if a's EOBB interected with boundary of G[a] then
        Find the neighbour cells base on a's motion angle
        foreach Agent b in the neighbour cells do
            if distance(a, b) < d then CollisionDetection(a, b)
        end
    end
end
```

5.2.5 Summary

The grid data structure has the advantage that it does not need to be updated, and an efficient hash map implementation can provide fast look-up. The drawback is that if the objects are of greatly varying sizes, the cells of the grid can be too large for the smaller objects while too small for the largest object, so the grid is both too fine and too coarse. Second, the hash map is quite time-consuming.

5.3 Quadtree

In this section we will overview the collision detection and avoidance by using quadtree. We will focus on two main aspects: updating the quadtree, since the object is moving all the time, and the neighbour search to reduce the number of collision queries. Finally, we present our implementation of quadtree.

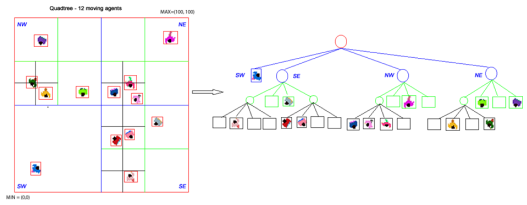

Figure 5.6: Construction of a quadtree, the scene is subdivided to four sub-square regions recursively until each node contains only one agent

5.3.1 Construction

To build a quadtree, the space is first divided into four sub-square regions (children nodes), namely the northeast (NE), northwest (NW), southwest (SW) and southeast (SE) according to direction. Fig. 5.6 depicts an example of quadtree data structure. Square region SW only includes one object, so will come under the "leaf node" category. There is no need to decompose it further; it will remain as a leaf of the quadtree. Other nodes still include many objects, so they must be recursively subdivided into four sub-quadrants, which form children of the corresponded node. The subdivision procedure is repeated continuously until either of the conditions mentioned below is satisfied.

1. The node is empty.

2. The number of objects in the quadrant is no more than a certain threshold number.

The Quadtrees structure is simple but unbalance when the objects are not uniformly distributed on the scene (Fig. 5.7). The high-density region of the tree is deeper than the low-density region of the tree.

5.3.2 Updated Strategy and Algorithms

There are two types of strategy to update quadtree. Because crowd simulation is a highly dynamic scene, the position of each agent is changing all the time; therefore the quadtree can be rebuilt at each frame. This strategy is

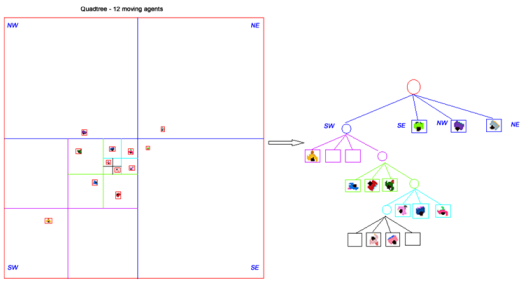

Figure 5.7: An unbalanced quadtree. Agents converged at middle of scene, most of nodes are empty, only one node needs to subdivide four times

simple to implement, but many times consumes for a large-scaled scenario. Second strategy updates the tree only when objects move out of current leaf node to other node, which includes two operations: traversal of the tree to find the leaf node which contain the object first, deleting the agent from the leaf node if it moves out; then, update node's structure (delete the leaf node if it is empty, or even delete its father node when it is also empty), then add the object into the new leaf node. The updating pseudo code is shown in algorithm 2.

5.3.3 Nearest Neighbour Search

We present a nearest neighbour search (NNS) algorithm for quadtree, so the neighbours of each agent can be queried efficiently. The basic idea of the algorithm is to traverse the quadtree to find agents who are near the query agent, and can potentially collide. The algorithm should get result by checking as few agents as possible. Principally, quadtree should allow extremely efficient implementations of NNS. Due to their regular partitioning of the search space and the high branching factor, coordinate queries are very fast [18]. Fig. 5.8 shows an example of NNS, where the grey area is the nodes which need to be searched. The NNS approach is summarized in algorithm 3.

Algorithm 2: Updating Qudatree

```
Input: Quadtree Q, Agents, the maximum number of agents allows in
       leaf node N
begin
    foreach Agent a in Agents do
        traverse Q to find the leaf node L which contain a
        if a's new location ∈ L then
            continue
        else
            delete a for L
            if L is empty then delete L
            hasChild == false
            foreach Child node C in L's father do
                if C is not empty then
                hasChild == true
                exit;
            end
            if !hasChild then delete L
            find the new leaf node L_new which contain a's new location
            adding a into L_new
            if total number of agents in L_new > N then subdivde L_new
            insert all agents into different sub-leaf nodes

        end
    end
end
```

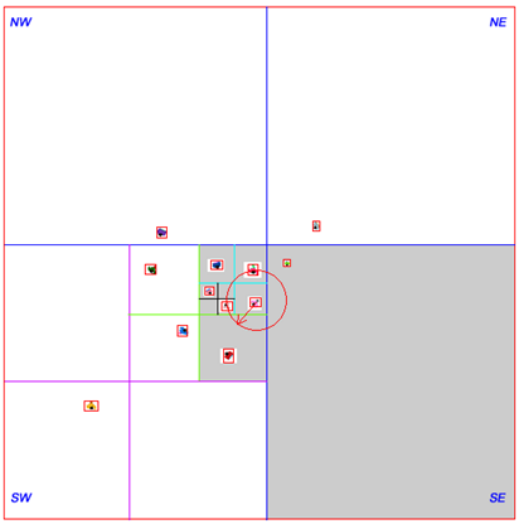

Figure 5.8: NNS in quadtree, only the agents in the grey colour region need to be check

Algorithm 3: NNS in Quadtree

```
Input: query point q, q's bounding box with maximal allowed
       distance d and the current node N
Output: Neighbours V
begin
    if N is LeafNode then
        foreach Agent a in N do
            if Distance(q, a) < d then V.add(a)
        end
    else
        sorting four child node of N based on distance to q
        foreach Child node C of N do
            get next closest child C
            if C is outside the bounding box then
                exit
            else
                NNS(q, d, C)
            end
        end
        Return V
    end
end
```

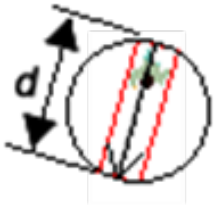

Figure 5.9: The region covered by the circle is the agent's potential collision area, the diameter of the circle is equal to the length of the large walking stride of agent

5.3.4 Implementation

Based on the definitions of the previous section, quadtree can be implemented easily. To simplify construction of quadtree, the agents is denoted as a particle with the agent's centre position, then inserts agents into quadtree by dividing the space into four uniform parts recursively. When an object intersects with y-direction splitting plane, the object is assigned to a west side node; if it intersects with x-direction splitting plane, it is assigned to north side node. Based on those rules, the objects can only be stored in one leaf node. After the quadtree is built, it is updated each frame by using algorithm 2, and the NNS approach in algorithm 3 is used to find the agent's neighbours. The d in NNS algorithm is defined by the length of the large walking stride of agent (Fig. 5.9).

5.3.5 Summary

The main benefits of quadtrees are that they subdivide space uniformly and are simple to implement. The regular subdivision of a quadtree has a fast indexing, which allows fast and easy traversing to a particular node directly. However, quadtrees also have a number of drawbacks. First of all, quadtrees are inflexible because of their fixed fan-out factor and space partitioning. Due to this, no direct control over the number of internal nodes is possible. Quadtree can thus be highly unbalanced (Fig. 5.7). Moreover, quadtrees are

complicated for nearest neighbour searches (NNS): NNS in a quadtree has to make proximity calculations to two split planes, sort the distances and select the appropriate sequence of traversal for every traversed node. In the next chapter, an evaluation will be given based on our experiment.

5.4 K-D tree

A k-d tree is a binary tree which divides a k-dimensional space hierarchically using a set of axis-aligned splitting planes, and it is a popular choice of acceleration structure in collision detection and ray-tracing. The art of using a k-d tree is summarized in Chapter 2. This section will introduce the concepts and terminology used in our research.

5.4.1 Construction

We consider a scene S contains N objects (Fig. 5.10). A k-d tree over S is a binary tree that recursively subdivides the space covered by S: the root corresponds to the axis-aligned bounding box (AABB) of S; interior nodes represent planes that recursively subdivide space perpendicular to the coordinate axis; and leaf nodes store references to all the agents overlapping the corresponding spatial. A k-d tree construction scheme is shown in Algorithm 4.

The splitting strategy directly influences how many traversal steps have to be performed. The traditional way is subdivide space uniformly(Fig. 5.10), which is easy to implement, and for the static scene also provides good performance. However, crowd simulation is a highly dynamic scene: pedestrians' positions always change and they are not uniformly distributed most of time, so the k-d tree can be unbalanced if we use this strategy (Fig.5.11a), as it would lead to the same problems encountered with quadtree where more collision queries need to be executed. To avoid this shortcoming, Median-cut is usually used, which adaptively moves the split plane to produce a balance k-d tree, which can make the situation more controlled. Balanced k-d trees can provide an equal amount of objects on each side of the splitting-plane and it usually have better performance than unbalanced trees. Fig. 5.11 compares balanced and unbalanced k-d tree, showing that the unbalanced

Algorithm 4: Recursive K-D tree build

Input: Agents A, scene S
Output: Node N
begin
  if *Terminate (A, S)* **then**
    return new leaf node(A)
  else
    p = FindPlane(A, S) `/* find a "good " plane p to split S */`
    (S_L, S_R) = split S with p
    $T_L = t \in T \mid (t \bigcap S_L) \neq \emptyset$
    $T_R = t \in \mathrm{T} \mid (t \bigcap S_R) \neq \emptyset$
    return new node(p RecBuild(T_L, S_L), RecBuild(T_R,S_R))
  end
end

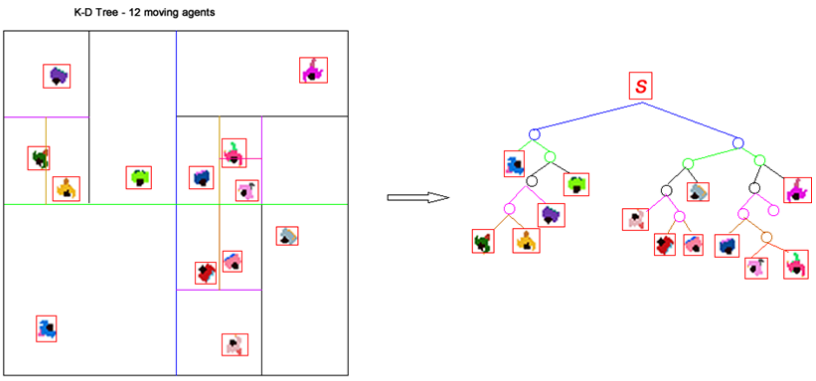

Figure 5.10: 2D scene S and the corresponding K-D tree,the middle of the axis is select for the split plane

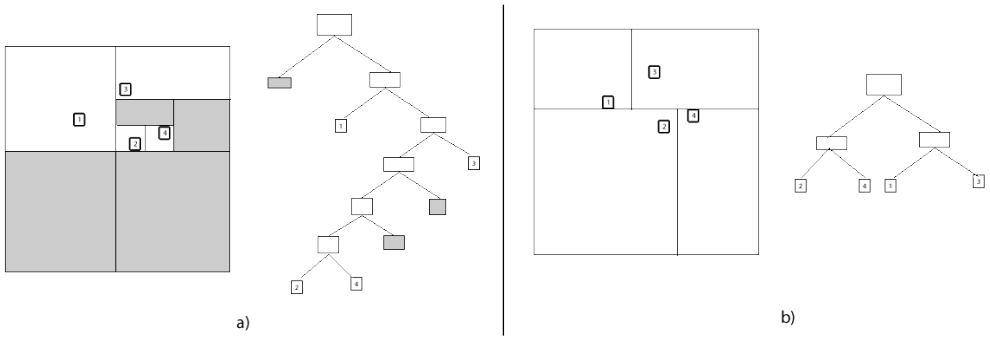

Figure 5.11: Balance and unbalance k-d tree. a) The middle of the axis is used, the unnecessary partitioning can be seen clearly in this graph. the grey regions are empty nodes, the depth of the tree is six. b) The median cut is used, only two levels in this balance tree

tree needs to traverse much deeper to find a particular agent than balance tree, and lot of unnecessary partitioning in unbalanced tree (Fig. 5.11a).

5.4.2 Quick Sorting

Since the median-cut is used, objects need to be sorted to find the median before building the k-d tree. Many sort algorithms are used in computer science. A Quick sort is used in our implementation (Fig. 5.12) and is easy and fast to build. On average, it makes O(n log n) comparisons to sort n items, and with an in-place partitioning algorithm, it only use O(log n) additional space. Due to sorting approach, the balanced k-d tree takes slightly longer time to construct than unbalanced k-d tree, though look up visits about one third fewer nodes on average.

5.4.3 Nearest Neighbour Search (NNS)

NNS in a k-d tree is simpler than in quadtree, because k-d is binary tree, sorting the distances and selecting the appropriate sequence of traversal for every traversed node is unnecessary, and a single proximity check can determine the order of traversal instantly.

Fig. 5.13 shows the procession of nearest neighbour searching. The grey nodes are visited when we query with the grey rectangle. The node marked

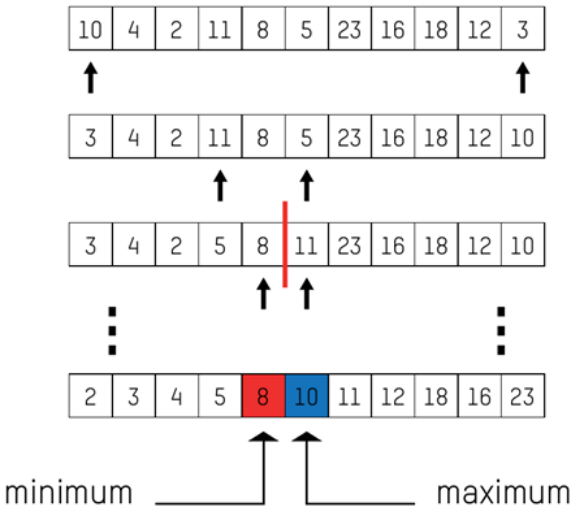

Figure 5.12: Quick sort: on average, makes O(n log n) comparisons to sort n items, and with an in-place partitioning algorithm, it only use O(log n) additional space

with a star corresponds to a region that is completely contained in the query rectangle, which is shown by the darker rectangle. Hence, the dark grey subtree rooted at this node is traversed and all points stored in it are reported. The other leaves visited correspond to regions that are only partially inside the query rectangle; hence the points stored in them must be tested for inclusion in the query range. This results in point P6 and P11 being reported, and points P3, P12 and P13 not being reported. The search algorithm does a recursive descent on the side of the splitting plane closest to the search agent, then explores alternative branches. Algorithm 5 gives an overview of this method. Initially, the heap H of size k holds the current candidates for the k nearest neighbour and allows fast check and insertion. The linear heap is used which has a complexity of O (k), which is faster with a small constant.

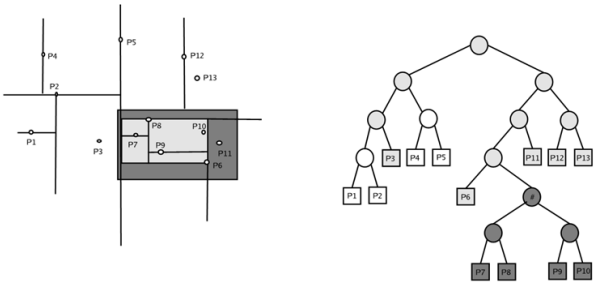

Figure 5.13: K-d trees nearest neighbour search (NNS), the objects in the grey colour region are all the potential colliding objects, the objects in dark grey region also need to be checked

Algorithm 5: NNS in k-d tree

```
Input: query poing q, node N, heap H, maximal squared allowed
       distance d
Output: heap H
begin
    if N is a leaf node then
        foreach Agent a in N do
            if Distance(q, a) < d then H.add(a)
        end
    else
        get split dimension s and split value v from N
        O = Distance(q[s], v)
        if O > 0 then
            NNS(q, rightChild(N), H, l)
            if ABS(O) ≤ l then NNS(q, leftChild(N), H, l)
        else
            NNS(q, leftChild(N), H, l)
            if ABS(O) ≤ l then NNS(q, rightChild(N), H, l)
        end
    end
end
```

5.4.4 Implementation

Same as quadtree, there are also two strategies to update the structure. They are: rebuild the whole tree, or just updated part of the tree. Different from the quadtrees implementation, we rebuild the k-d tree each time instead of just update the tree. Because median-cutting is used to choose the splitting plane in the k-d tree, after the position of an agent is changed, the splitting plane also changes, and the tree must be rebuilt again. The algorithm 4 is used to construct our k-d tree in each simulation time, and the good splitting plane is selected by using median-cutting, and the quick sort is used to get the median value for each axis. Finally the longest axis always is used to divide scene. The terminate recursively criteria is either the number of objects falls below a certain threshold t, or until the subdivision depth exceeds a certain maximum depth d (*e.g. 5.3*).Where D(S) is the current subdivision depth.

$$Terminate(T, S) =\mid T \mid \leq t \vee D(S) \geq d \tag{5.3}$$

Similar to quadtree, the agents in a k-d tree are also denoted as particles with the position of the centre of the pedestrian (Fig. 5.14). We subdivide the scene, and if the objects position is on the splitting plane, it is always assigned to the left node of the tree. Therefore, each object can only be in one leaf node.

Finally, the Algorithm 5 is used to find the potential collided agents and culling off the unnecessary nodes. Similar to implementation of a quadtree, the query range is equal to the length of the large walking stride of the agent (Fig. 5.9).

5.4.5 Summary

In comparison with a quadtree, the order of traversal in a k-d tree is instantly determined by a single proximity check, thereby avoiding unnecessary computations, if nodes need not be visited in NNS. The k-d tree is a balanced tree, even if the agents are not uniformly distributed, and the neighbour searching can still be efficient. The drawbacks are because the median-cutting is used, the agent's positions need to be sorted each time, and the k-d tree needs to store more information to identify sub-regions since the longest axes are used

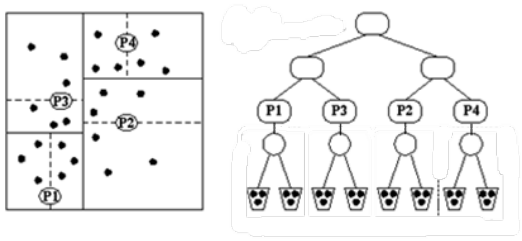

Figure 5.14: The Agents are represented by particles in k-d tree. The agent always assigned to the left node if it intersect with splitting plane. Therefore, the object can only be in one leaf node

for splitting plane and partitioning is not uniformed.

5.5 Bounding Interval Hierarchy

Bounding interval hierarchy (BIH) is similar to that of k-d trees and can be fast and easy to implement in ray tracing. The backgrounds have been introduced in Chapter 2. In this section, we will overview the construction and traversal approaches by following our implementation.

5.5.1 Construction

The data structure of BIHs consists of two types of nodes: the first is internal node, which contains two references to both left and right child nodes; the second is the leaf nodes, which hold a reference to the beginning of an object list. This is similar to the k-d tree data structure but each internal node in a BIH has two parallel splitting planes (k-d tree only has one splitting plane) that are orthogonal to one of the tree coordinate axes. The objects in the scene are divided into left and right child nodes by those two planes (Fig. 5.15). The BIH's nodes are consisted by the two clipping planes and a pointer to a pair of children.

Construction of a BIH is simple. Fig. 5.16 shows the approach, assuming that the longest axis in a scene is in the y-direction, where the position

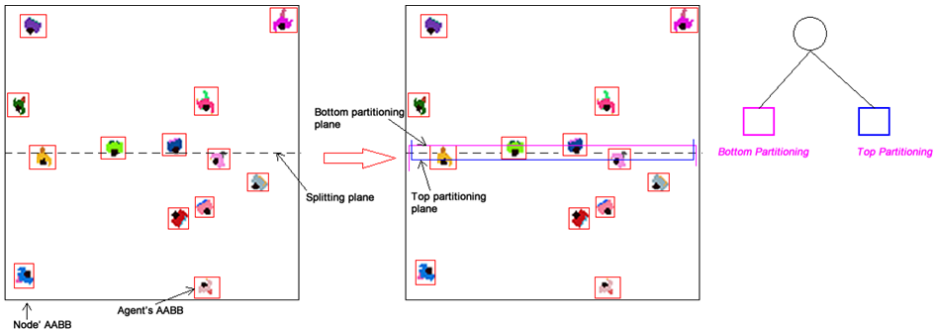

Figure 5.15: BIHs: two clipping planes are used, so the agents can tightly fitted into each leaf node

of the splitting plane is x_0. The AABBs of the objects within the node's volume are then sorted along this axis. The objects whose AABBs have all x-coordinates less than or equal to x_0 are assigned to the left child. AABBs that are entirely on the right of the splitting plane are assigned to the right child. Objects whose AABBs intersect the splitting plane are classified as belonging to the left or right child depending on which side of the splitting plane the AABBs have maximum overlap. The left partitioning plane is then defined using the maximum value of the x-coordinates of the AABBs belonging to the left child, and the right plane is defined using the minimum value of the x-coordinates of the AABBs belonging to the right. The process continues by splitting each child node along the longest axis and defining two partitioning planes along that axis. A node containing only a single object is not subdivided further. The scheme is show in Algorithm 6.

5.5.2 Nearest Neighbour Search

Storage of a BIH is similar to a BVH and therefore the empty space between two non-overlapping child nodes is possible to eliminate when doing the neighbour searching. For example, in Fig. 5.17 the object on the left side is intersected with the medium splitting plane, but after using two splitting planes in the BIH, the object is only contained within left node. The BIH is also benefiting from the k-d tree, since BIH is binary tree, and the nodes in BIH are spatial distributed, the fast nearest neighbour searching algorithm

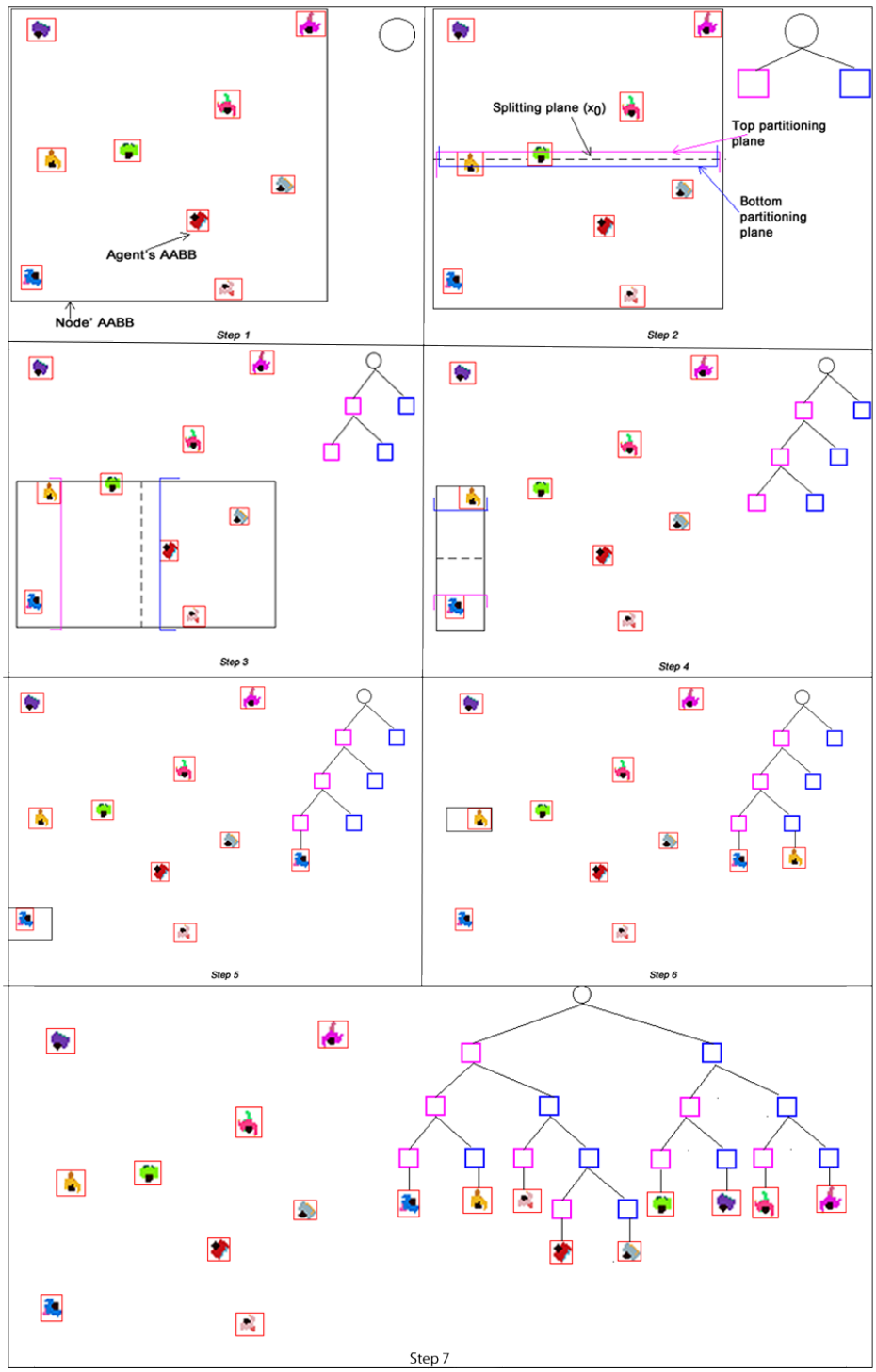

Figure 5.16: Approach of building a BIH. The split plane candidates x_0 is selected by middle of axis, which is determined by the global bounding box of the scene. Two clipping planes are selected by compare the maximum and minimum value of bounding box of agents.

Algorithm 6: BIH Construction

```
Input: Agents A, Node N, Scene S
Output: Node N
begin
    if Terminate (A, S) then
        return new leaf node(A)
    else
        p = S.middle  /* middle point of S is split plane */
        foreach Agent a in A do
            if a. midposition < p then
                p_l = a.OBB.max   /* maximum value of a's OBB */
            else
                p_r = a.OBB.max   /* mimimum value of a's OBB */
            end
        end
        S_L = split S with p_l
        S_R = split S with p_r
        T_L = t ∈ T | (t ⋂ S_L) ≠ ∅
        T_R = t ∈ T | (t ⋂ S_R) ≠ ∅
        new node(p_l, p_r, RecBuild(T_L, S_L), RecBuild(T_R,S_R))
    end
end
```

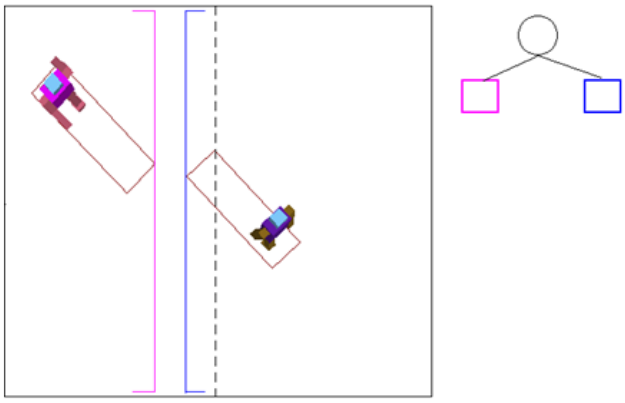

Figure 5.17: Eliminating empty space between two non-overlapping nodes, and avoid the agent are intersect with two nodes

in k-d tree can also be implemented into BIH.

5.5.3 Implementation

BIH data structure records objects into their nodes, instead of just recording the particles. In our implementation, EOBB is used to represent the agents. An example of construction of BIH is shown in Fig.5.18. The minimum and maximum values of EOBB are used when choosing the two splitting planes.

5.5.4 Summary

BIH usually needs to traverse a lower number of neighbours than a k-d tree when looking for potential colliding neighbours, because by using two partitioning planes, a bounding interval hierarchy is able to classify each object in a node volume uniquely as either left or right, without the need for placing an object that overlaps the splitting plane in both child nodes. The construction of BIH shows that the empty space can be eliminated. The interval hierarchy thus provides a hierarchy of axis aligned bounding volumes and also a spatial ordering similar to that of the k-d tree, which should provide fast NNS. The drawback of a BIH is the extra computation needed to sort the object, and that the maximum and minimum values of each EOBB need to be calculated first, so that the two splitting planes can

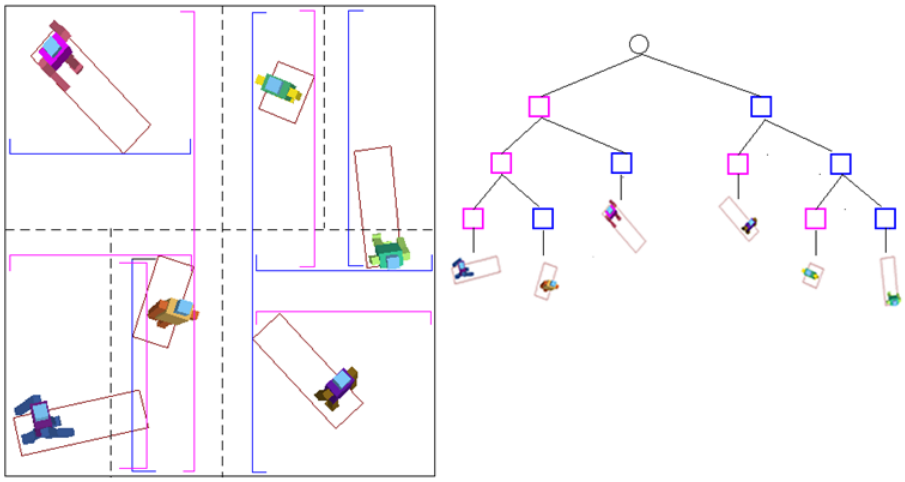

Figure 5.18: Implementation of BIH. The EOBB of the agents are used in BIH, and the maximum value and the minimum value of EOBB is calculated to choose the two clipping planes, so each object is only in one leaf node, and the leaf node can tightly contain the agents.

be found.

Chapter VI

Results and Evaluation

This chapter summarizes the results of the proposed approach. In Section 6.1, the extended oriented bounding box (EOBB) algorithm will be tested in different scenarios. The aim is to subjectively evaluate the experimental results to determine the performance of collision avoidance and motion update algorithms. Later, the four spatial partitioning data structures are implemented for allowing large-scaled crowds to run in real-time. The comparative analysis between each of the data structures is discussed in Section 6.2. The next section, section 6.3, provides the evaluation of evacuation times under several scenarios measured against our simulation model. The comparative analysis between different crowd simulations models is given in Section 6.4. Finally, Section 6.5 provides the detailed information about the hardware configuration used to collect the final simulation results along with details of the development environment.

6.1 Experimental Results: Motion Planning with EOBBs and Obstacle Avoidance

In this section, the E1 and E2 (detailed in Chapter 3.3) are used for counterflow within a corridor and a highly constrained environment with bottlenecks respectively.

6.1.1 Motion Planning with EOBBs

The results when using E1 are discussed in this sub-section. Fig. 6.1 1 shows the initial configuration of the simulation. In this scenario, 120 agents are dispersed onto the corridor. The corridors can accommodate no more than 40 pedestrians walking in line. Colours are used to differentiate between the two groups: the left side crowd is in blue and right side is illustrated in red. Motion planning using EOBBs and without EOBBs are implemented. The

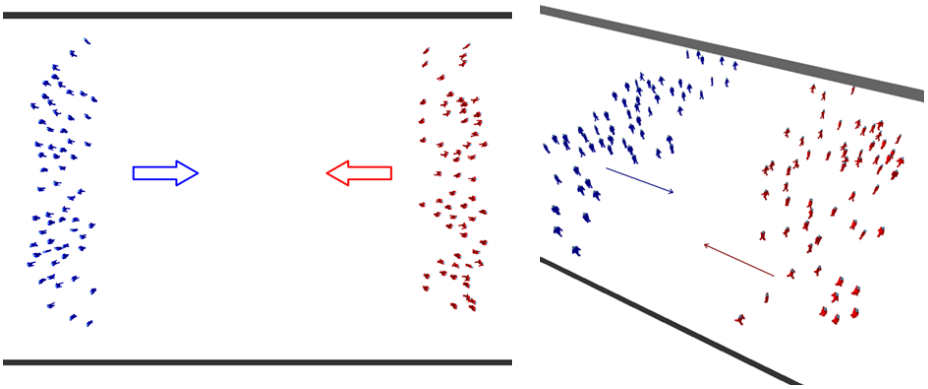

Figure 6.1: Environment 1 (different viewpoints): Two crowds moving in opposite directions (wide corridor)

experimental results are explained in this section.

A normal collision avoidance method is implemented initially. In every simulation step, the new position of each pedestrian is calculated based on his velocity. Collision detection is executed and the pedestrian's position is updated to the newly elevated position,as long as the pedestrian does not collide with others in the newly estimated position. If any colliding happen, the pedestrian will not be moved and the current direction will be updated by adding an offset angle. This method is simple to implement, but has a major drawback: penetration and deadlock can happen in case of high density crowd, due to the limitation that in each time frame, the direction of agents can be updated once only. There are chances that two objects will collide when updating with the new direction. Fig. 6.2 shows an example situation of a deadlock condition. To address these issues, we introduced an EOBB structure. Firstly, EOBB can help avoid collision. Secondly, a new motion planning method is estimated by using three EOBBs to resolve deadlocked conditions (detailed in Chapter 4.1 and 4.2). The results are explained and analysed below. Note that flow fields are used to visualise the performance of the algorithms.

Results show that the pedestrians in two groups can avoid collision and reach the goal perfectly by using our motion planning algorithm. The different stages of the simulation are shown in Fig. 6.3a showing two crowds

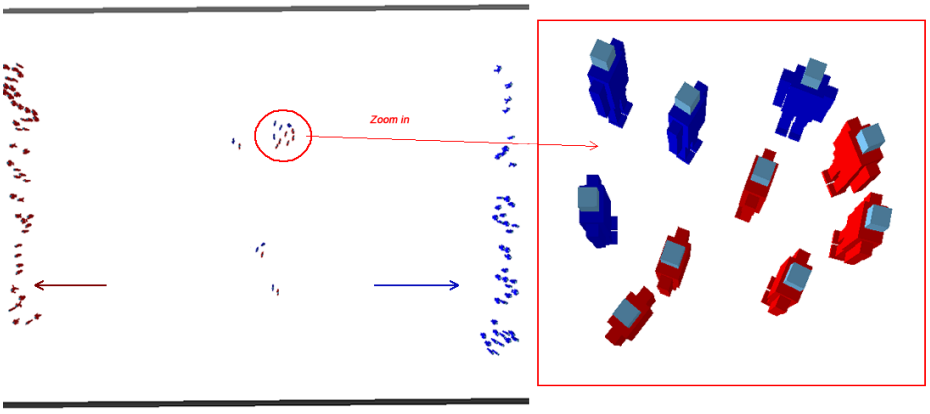

Figure 6.2: Screenshot of simulation: deadlock condition in the middle of the scene, the right side graph shows the zoomed in of deadlock situation

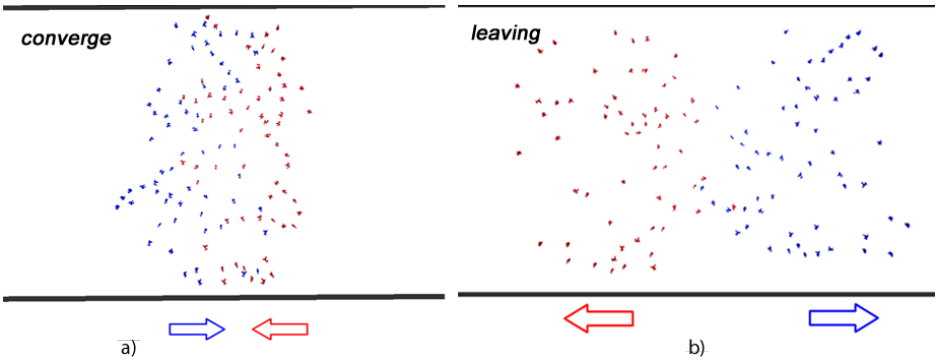

Figure 6.3: Screen-shot of simulation: a)crowds converging in middle of corridor, b) crowds leaving.

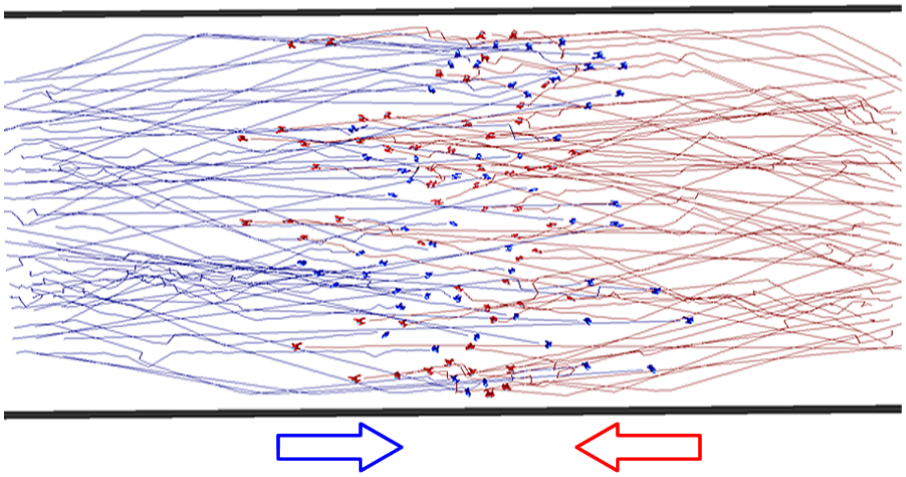

Figure 6.4: Flow field of the start state of convergence

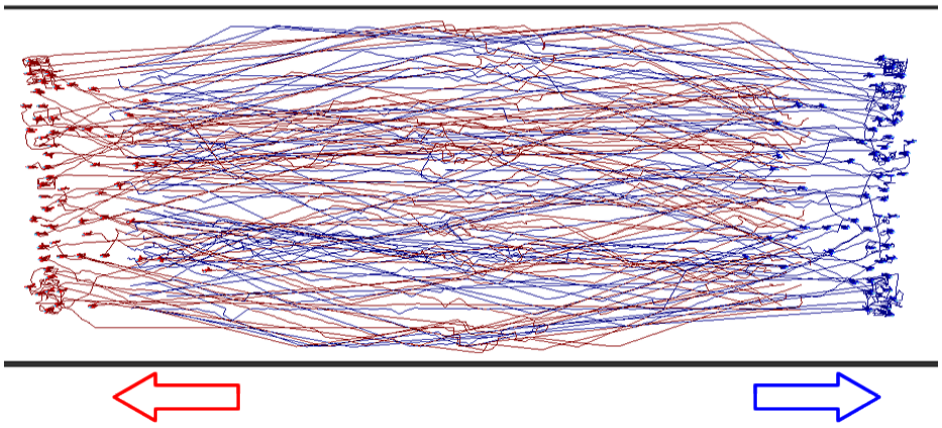

Figure 6.5: Flow field showing crowd motion towards targets in a scene

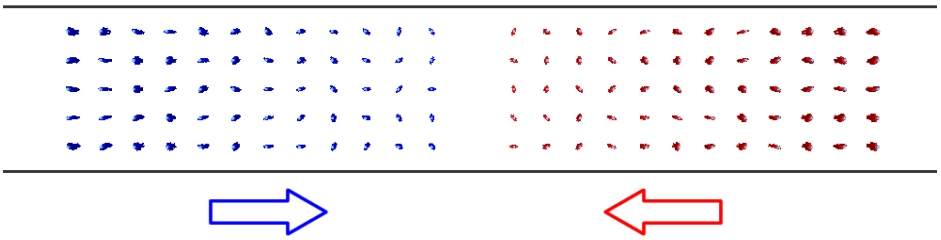

Figure 6.6: Two crowds moving in opposite directions.

meeting in the middle of the corridor, at the middle of the simulation. Fig. 6.3b shows the final stage of simulation where the crowd has reached the destination. The flow field for this scenario is given in Fig. 6.4 and Fig. 6.5,which maps the motion of the two groups.The directions keep changing constantly to find a free path and subsequent target tracking can be clearly seen at several locations in the flow field. Our motion planning algorithm can guide pedestrians to find free paths perfectly, and there are no congestions between the two groups when they meet in the middle of the scene. Another major advantage is that the deadlock situation is always avoided. In this experiment, pedestrian distribution is scattered. Note that to increase the density of the crowds in the next experiment (Fig. 6.6), the width of the corridor was reduced.

6.1.2 High-density Crowds

In this environment, the width of corridor allows only 12 pedestrians to walk in a line (Fig. 6.6 and Fig. 6.7). Other settings are the same as configured for the previous experiment.

Two important observations from the simulation results are firstly, that the our algorithm runs well, even in this high density environment. Secondly, compared with flow field in Fig. 6.5, the directional changes near areas of congestion can be clearly seen in Fig. 6.8 and Fig. 6.9 due to the high density.

To evaluate the motion planning algorithm further under extensive con-

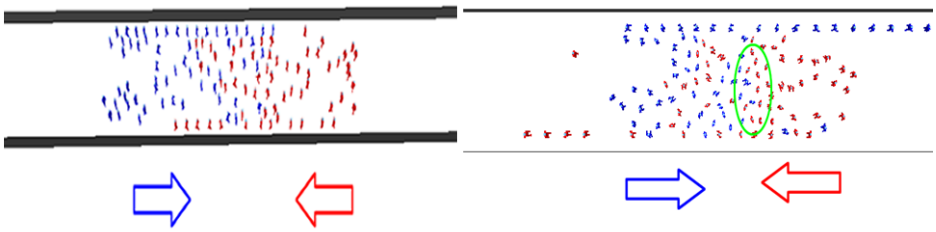

Figure 6.7: Two crowds converging in the middle of corridor. The green circle indicates congestion; the two crowds cannot pass through the middle area.

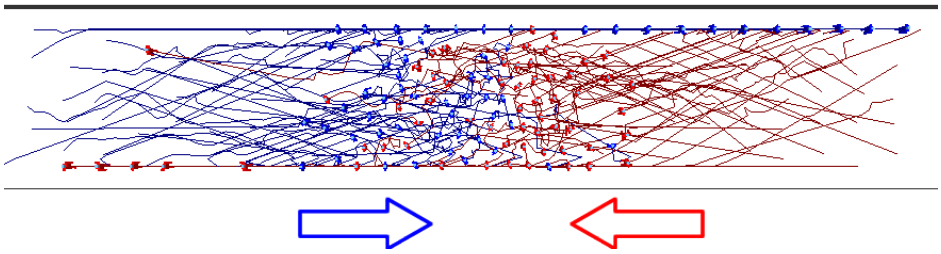

Figure 6.8: Flow field of two crowds converging. The graph shows the direction start changes near the congestion area at middle of the corridor.

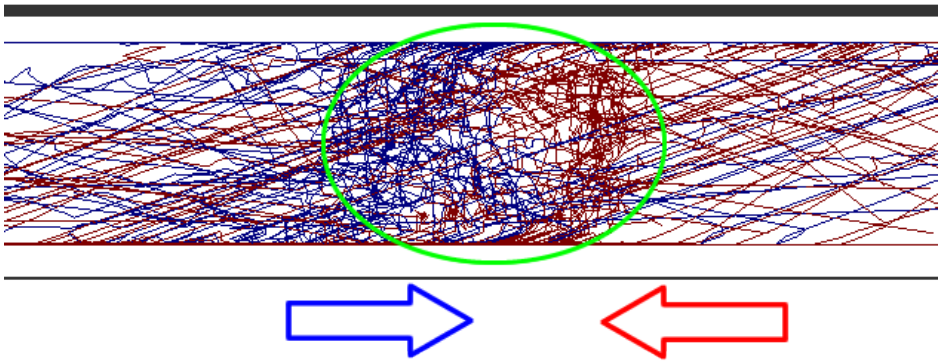

Figure 6.9: Flow field of two crowds moving in opposite directions in a narrow corridor. The directional changes near areas of congestion and subsequent target tracking (circled in green) can be seen clearly in the middle of the corridor in this flow field.

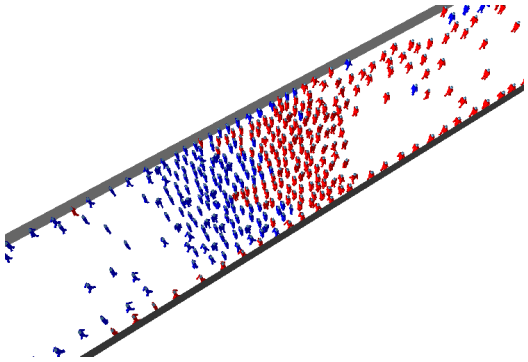

Figure 6.10: Two crowds converging in the middle of the corridor. Due to congestion, only a few agents can move

ditions, we increased the crowd size from 100 to 1,000 agents, adding crowd in steps of 100. We investigate the bottlenecks involved in such a situation. The total travelling times for different crowd size are measured. Congestion starts occurring when the crowd size is 500. This could be due to the narrow width of the corridor (Fig. 6.10). The congestion increases many-fold, owing to the increasing crowd size. When the crowd size reached 1,000, most of the pedestrians converged in the middle of the corridor stop moving, there is a deadlock situation. To avoid this problem, a separating median wall can be added in the middle of the corridor. The median wall separates the two directions of movement (Fig. 6.11). Effectively, the total walking area is decreased by introducing the wall. Another observation is that the total travelling time is increased dramatically,as shown in the graph in Fig. 6.12. The wall separates the two crowds onto two sides, each moving in a different direction on a different path, and there is no congestion during the simulation. The total travelling time is shown as a graph in Fig. 6.12. The graph also indicates that when pedestrians are less than 500, the wall is not needed.

So far, the collision detection is only considered between agents. In the next experiment, we increased the complexity of this environment by adding three obstacles in the middle of the scene (Fig. 6.14), a situation where we could have collisions between agents and obstacles. This set up also allows for testing and evaluating the obstacle avoidance algorithm. The results are discussed and analysed in the following sections.

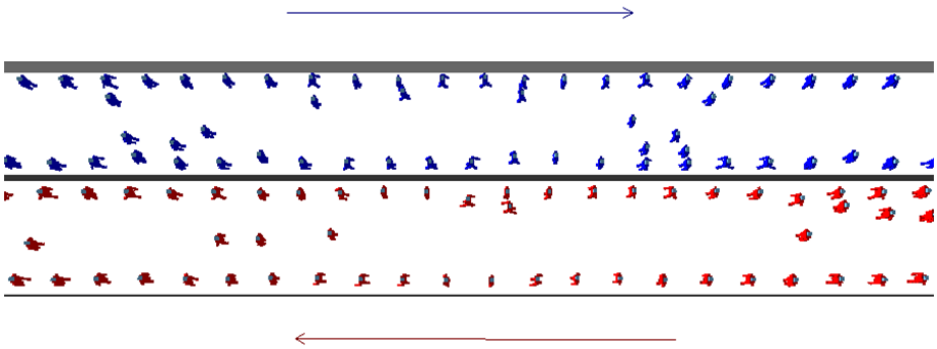

Figure 6.11: Corridor with separating wall. The two crowds can move, in only one direction on each side.

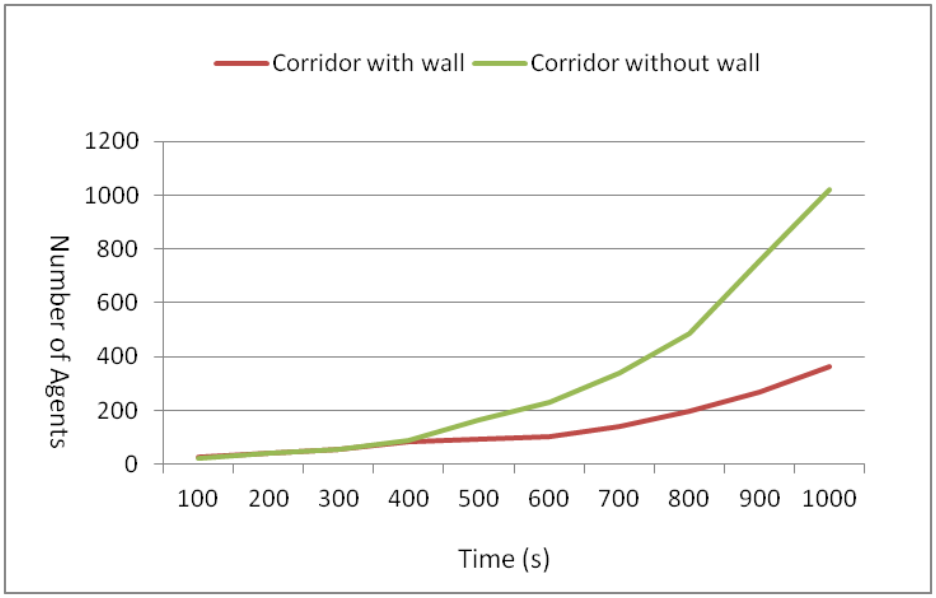

Figure 6.12: Total travelling time in the corridor with and without the separating wall

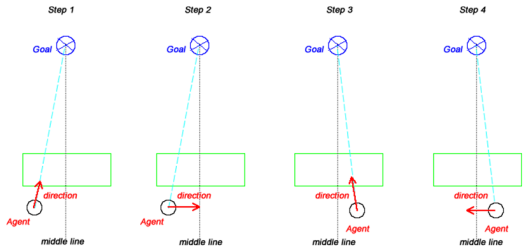

Figure 6.13: Deadlock situation between agent and obstacle. Agent keeps changing direction, and can never pass the obstacle.

6.1.3 Obstacle Avoidance

Fig. 6.13 illustrates the deadlock situation when an agent tries to pass an obstacle. In Step 1, the agent will collide with the obstacle if it moves in the current direction, so the agent has to change its direction. Based on the shortest path calculation, the agent turns his direction to the right and keeps moving(Step 2). In each simulation step, the agent will detect chances for collision with an obstacle until he finds a free path to move. A deadlock may happen at Step 3, after the agent passes the middle line. Based on the shortest path estimation, the agent's next direction will change to the right instead of the left(Step 4). To avoid oscillations, the obstacle avoidance method, as discussed in Chapter 4.2, is implemented here. Each wall of obstacle has a direction assigned to it, and based on this direction, the *e.q.4.1* in Chapter 4.2 is used to calculate the new direction and speed of pedestrians where there could be a possible collision.

The Fig.6.14 - Fig. 6.17 show that our motion plan algorithms still works well, even in a complex environment. The results also prove that pedestrians can pass obstacles smoothly and without colliding. Fig. 6.16 shows that when agents come closer to an obstacle, agents at the top side move up and the agents at the bottom turn down, based on the direction of the wall. Fig. 6.17 also shows the two crowds can pass the rectangle obstacle smoothly by following the obstacle's directions.

Our EOBB structure is used in all previous experiments. The experimental results show that our EOBB structure is very effectual in collision avoidance. Fig. 6.19shows the details of implementation of the EOBB struc-

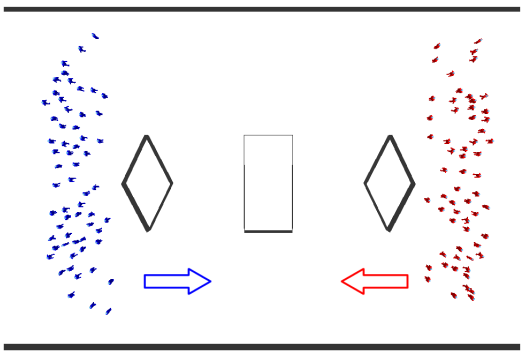

Figure 6.14: Two crowds moving in opposite directions. Three obstacles are located in the middle of the corridor.

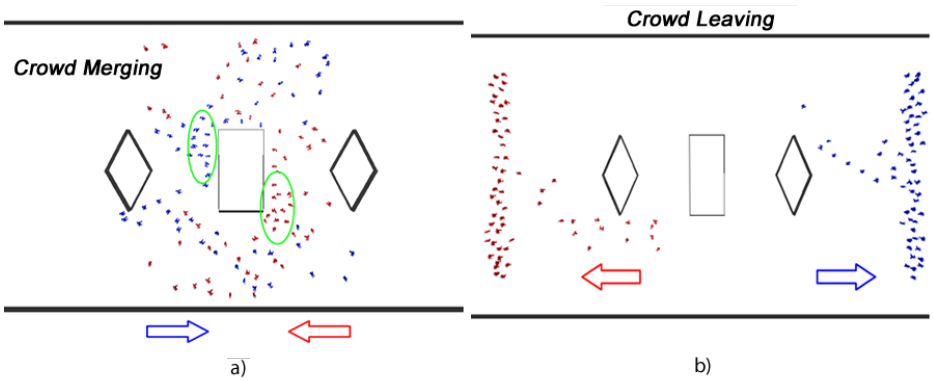

Figure 6.15: a) Two crowds moving in opposite directions start converging. The two green circles shows agents' ways being blocked by an obstacle. b) Crowd reached their destination without any deadlock.

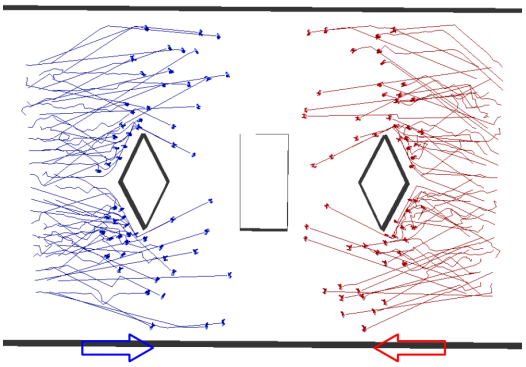

Figure 6.16: Flow field of start stage of experiment. Agents start changing their direction after coming close to the first obstacle.

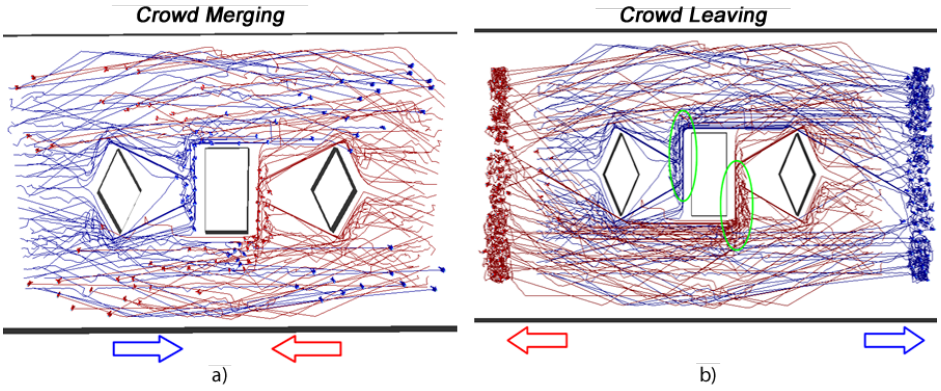

Figure 6.17: Flow field: a)In half the simulation time, crowds can avoid the obstacle perfectly by following the direction of the obstacle's wall. The right side crowd's direction turns top side and the left side crowd's direction turns bottom side. b) Flow field of final stage of experiment. The green circles show agents passing obstacles smoothly and subsequently moving on to the target.

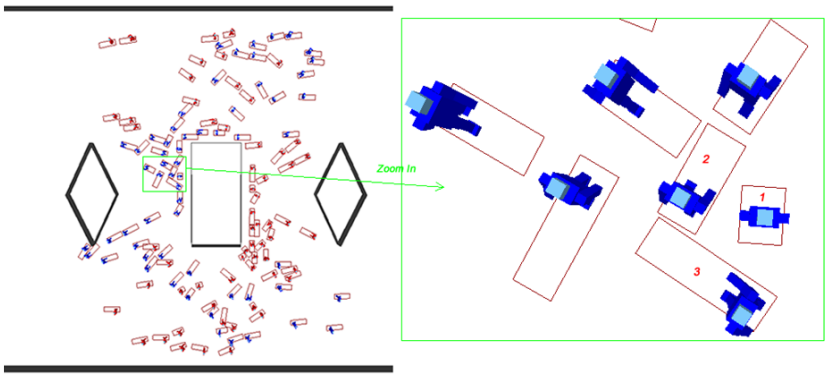

Figure 6.18: Screenshot of crowd simulation with EOBB. Agent number 1 has stopped moving, agent number 2 can only move with a slow walk-cycle, and agent number 3 is free and can move with a brisk walk-cycle.

ture. In the left image, each pedestrian is shown, bounded by EOBB with their own maximum stride length. The right side image is a zoom-in of a particular part of the scene with numbers indicating the three types of walking states. *Number 1* represents a pedestrian who has stopped walking, having an obstacle on his left and three other pedestrians surrounding him. Apparently there is no space for him to move. Therefore, his stride length becomes / is set to zero. Based on the same assumptions and observations, *Number 2* has limited space to move, so a slow walk-cycle is assigned to him. Finally, *Number* 3 has free moving space and will be assigned a brisk walk-cycle.

The next section will discuss how the angle β in three EOBBs affects the performance of crowd simulation.

6.1.4 Highly Constrained Environment (E2)

The environment is highly constrained, with an obstacle that limits the movement of pedestrians to a small region. E2 is used for this experiment. 200 pedestrians are initialized on the bottom of the scene and the destination is the top of the scene (Fig. 6.19). Different colours are used for identifying each of the pedestrians. We test three EOBBs (Fig. 6.20) with various angles β (30 to 120 degrees). The results and analysis will be discussed in later

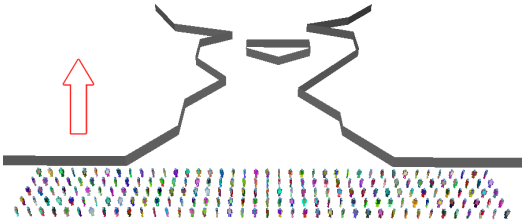

Figure 6.19: Highly constrained environment with 200 pedestrians.

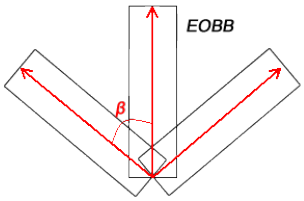

Figure 6.20: Three EOBBs with angle β.

sections.

Fig. 6.21 demonstrates that when the angle is 80 degrees we get the fastest travelling time. The figure also shows that we got worst performance when the angle was too small or too large. This is because when the angle is too small, the agents movement / directions are limited and they follow the agent in front of them when the crowd density is high. In contrast, when the angle is larger than 90 degrees, the agents can find a free path than in a small angle situation. There is also a possibility that when agents change the direction more than 90 degrees, they will move backwards. Therefore,

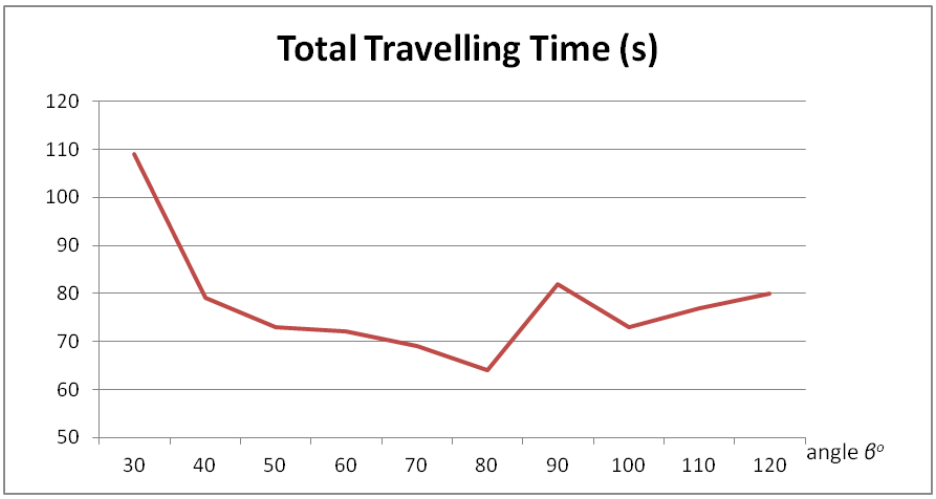

Figure 6.21: Total travelling time based on three EOBBs with different various angles.

in our experiments we have always used 80 degrees. The results indicate it is the best performance that can be simulated. In the following pages, we give the screenshots of different stages of simulation in a highly constrained environment.

The figures depict the motion of the 200 pedestrians in a clear and concise manner. In Fig. 6.22, it can be observed that pedestrians attempt to walk in a particular direction (indicated by green arrow) and avoid the right side because of the large corner (red circle). Fig. 6.23 shows the most congested area on the left side (green circle).The path between the left top corner and the obstacle is very narrow and limits the pedestrians' movement. It also shows the working of the obstacle avoidance scheme which prevents motion in the close vicinity of walls.

All these experiments prove that our algorithm can be used in both scattered as well as dense situations, and runs effectively even in complex environments. Until now, all experiments have been carried out using only a small group of pedestrians. To achieve a large-scale crowd simulation in real-time, acceleration data structures must be used. In the next section, the comparison between four spatial partitioning data structures is given.

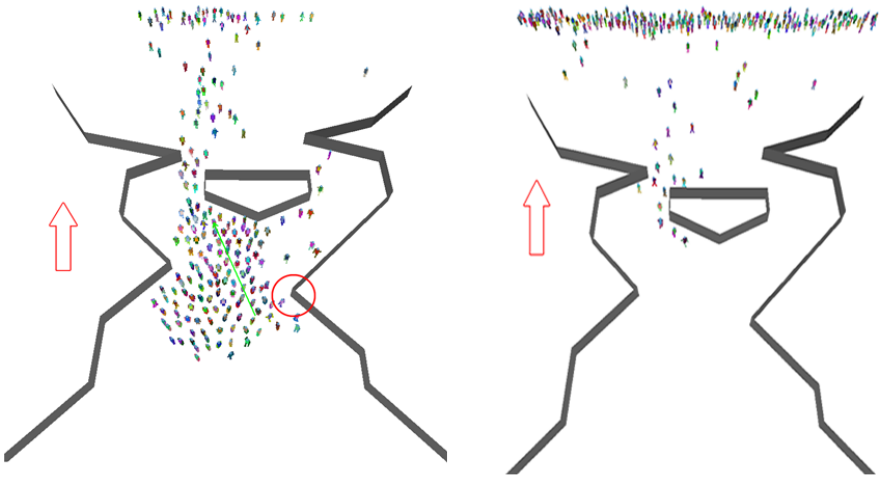

Figure 6.22: Highly constrained environment with 200 pedestrians. Pedestrians are more likely to travel along with left path.

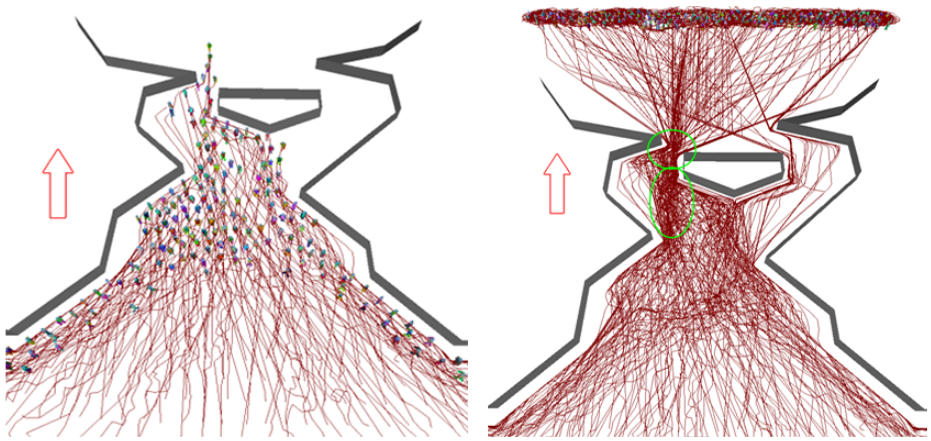

Figure 6.23: Flow field showing movements / path in the highly constrained environment. a) In half the simulation time, crowd move to target smoothly. b)Pedestrians move to the target without any colliding. Green circles point out the bottlenecks.

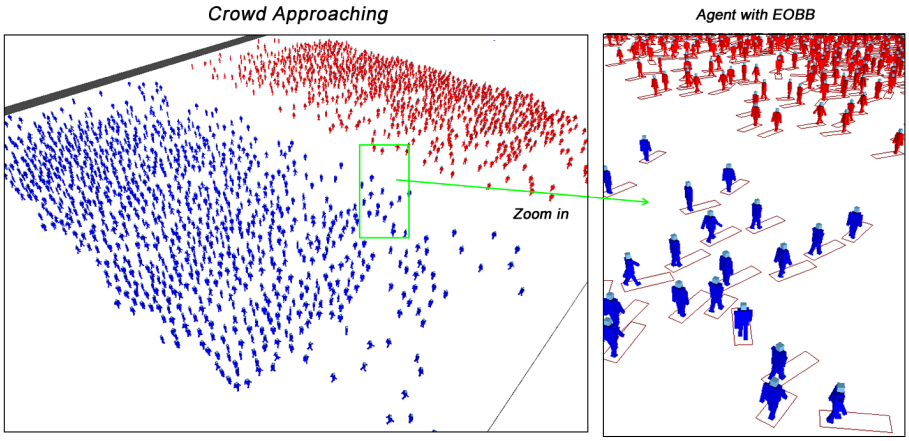

Figure 6.24: Crowd simulation with 2,000 agents. Each agent is bounded by an EOBB of a dynamically changing stride length.

6.2 Results and Evaluation: Acceleration Data Structures

For the purpose of experimental analysis, the environment E5 is used, which can contain a large number of crowd members. A typical scenario, consisting of two large groups of people moving in opposite directions, are shown in Figs 6.24, 6.25, and 6.26. Fig. 6.24 shows the initial configuration of the simulation. The groups meet in the middle of the scene at a later stage in the simulation (Fig. 6.25) and finally reach the destination (Fig. 6.26). In each figure, a small region is enlarged to show clearly the crowd members and their corresponding EOBBs with different stride lengths. Each scene can have increasing levels of complexity based on the behaviour models and path planning algorithms used. One commonly observed pitfall is that poorly designed algorithms can make inefficient memory requests. We implemented all four data structures described earlier and the performance of each case is evaluated with respect to increasing crowd size. The experimental results are discussed and explained in later sections.

6.2.1 Cell Size in Gird

The cell size is an important parameter in the design of grid data structure. In our experiments, the cell sizes are evaluated for four different scenarios:

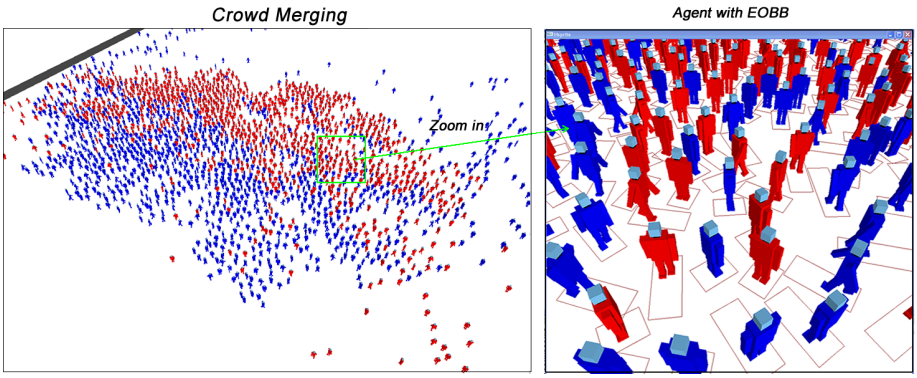

Figure 6.25: Crowd simulation with 2,000 agents. Two groups of agents start converging in the middle.

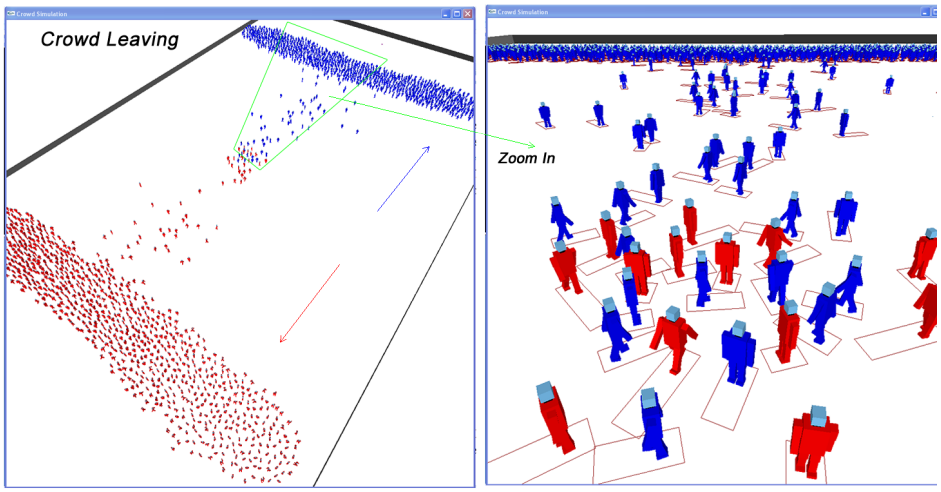

Figure 6.26: Crowd simulation with 2,000 agents, with a few agents walking to the destination while the rest have already reached the destination.

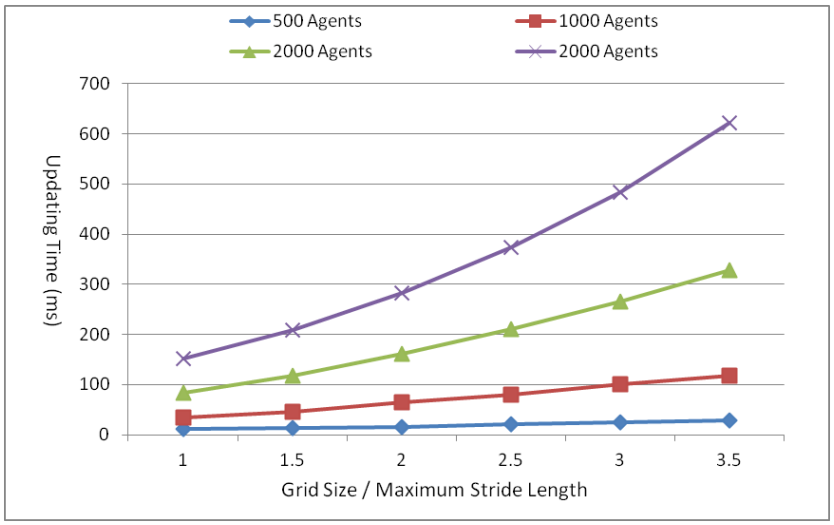

Figure 6.27: Updating time *vs.* cell size for the grid structure.

500, 1,000, 2,000 and 4,000 agents, respectively. In each scenario, different sizes of cells are tested. First, the size of a cell is set to the maximum stride length of each agent. Then the size of the cell is increased and the time required for updating is measured. The results are shown in Fig. 6.27. The best performance is obtained when the cell size just fits the maximum stride length of agent. Also, it has been observed that performance is dependent on cell size, particularly in cases of high density crowds. The probability of colliding with neighbours is calculated by using neighbour cells. If the cell size is larger, then the agents have more chances of staying in the same cell, so the number of potentially colliding agents is increased. If the crowd is scattered, each agent is far away from any others, so if the cell size is larger, there is a chance that a cell may contain more than one agent. In contrast, the high density crowd scene has more agents staying in the same cell when the cell size is larger.

6.2.2 Number of Agents in Leaf Node

The performance of tree structure is highly dependent on the levels of the tree and the number of agents in leaf node. In this section, we will describe the running of the simulation using varying number of agents in each leaf node

Figure 6.28: Updating time *vs.* the maximum percentage of agents contained in a leaf node for a quadtree.

for each of the tree data structures. The comparisons between each case are discussed in detail. As in the previous experiments, four different scenarios are used, each having 500, 1,000, 2,000 and 4,000-agents, respectively.

In Fig. 6.28, Fig. 6.29, and Fig. 6.30, we provide experimental results using quadtree, k-d tree and BIH to find the variation in performance with the maximum number of objects stored in each leaf node. The results indicate that when the objects are less than 1,000 in the simulation, the number of objects in each of the leaf nodes does not impact the performance of the simulation in any serious manner. But when objects' number are more than 1,000 and the objects in each leaf node are 1% of the total number of objects, both quadtree and k-d tree performed best. For BIH to perform well, it is possible to sub-divide the space until only one object is in each leaf node. The best performance was observed when the number of objects contained in each leaf node is 10.

6.2.3 Comparative Analysis between Four Data Structures

Based on previous experimental results, we set up optimized parameters for each data structure. Simulations were conducted with different crowd sizes. Initially we started at 500 agents, with increments of 1,000 in each simulation, until the numbers of agents reach 10,000. The time taken for updating for

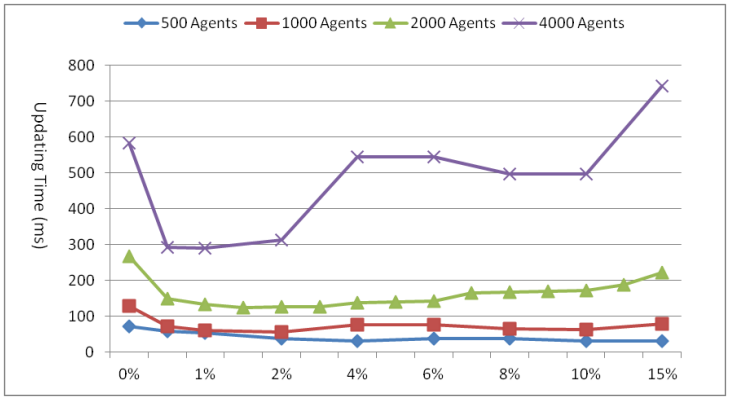

Figure 6.29: Updating time *vs.* the maximum percentage of agents contained in a leaf node for a K-d tree

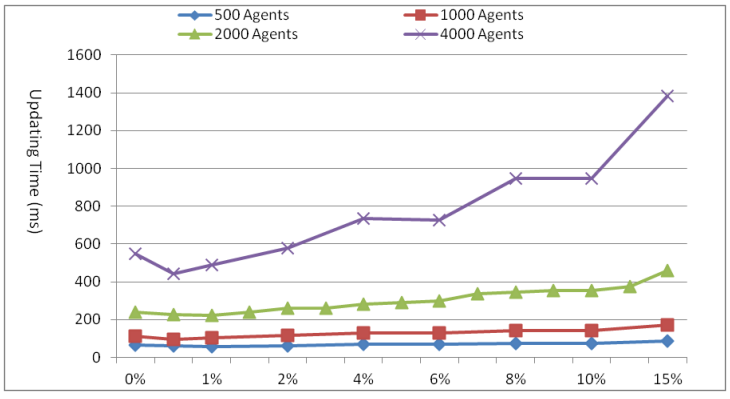

Figure 6.30: Updating time *vs.* the maximum percentage of agents contained in a leaf node for a BIH

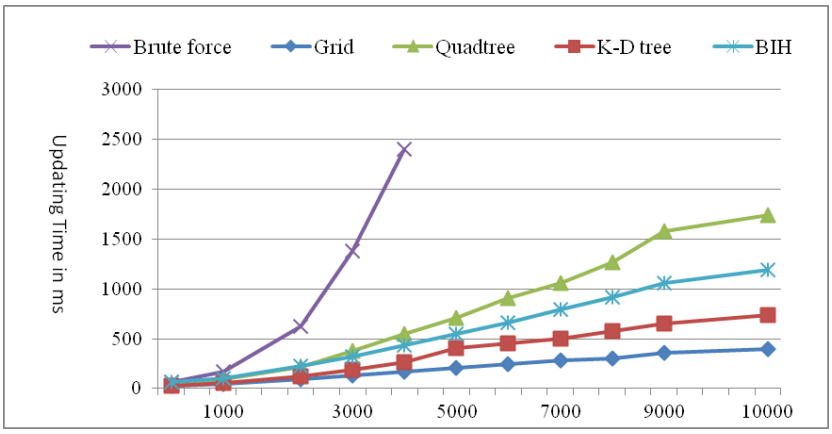

Figure 6.31: Updating time *vs.* number of agents (grid, k-d tree, quadtree, and BIH)

each of the data structures is measured. Screenshots from the 2,000-agent scenario are illustrated in Fig. 6.24, 6.25 and 6.26 for visualization purposes.

The times taken for updating each of the grid, k-d tree, quadtree and BIH algorithms are presented as a graph in Fig. 6.31. The graph shows that grid structure gives the best performance. Even when the number of agents in the scene is 10,000, the time taken for updates using the grid structure is less than 500ms. We also noticed that the grid structure did not provide a significant improvement over the brute-force method when the number of agents was less than 1,000.

6.2.4 Summary

The simulation results show that combining an EOBB algorithm with the grid data structure gives the best performance for large-scaled crowd simulation (Fig. 6.32). The advantages are firstly, that the EOBB reduces the neighbouring cells check from nine to four. All agents in our simulation are of the same size and objects keep moving continually at all times. The grid data structure does not need to be rebuilt after it is compiled initially. The O(n) updating and deletion time can be achieved by using a hash table, and the O(n) traverse time makes the neighbour search faster than the other tree structures. For example, the k-d trees minimum traverse time is log(n) for

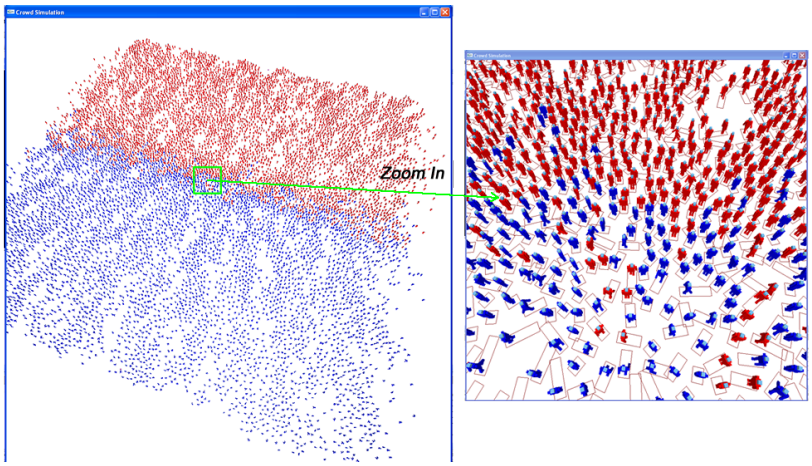

Figure 6.32: crowd simulation with 10000 pedestrians in real-time, grid data structure is used

each agent.

The k-d tree has better performance than quadtree and BIH. The objects are not uniformly distributed at most of the times during simulation, so quadtree can thus be highly imbalanced, and navigate deeper than necessary. Moreover, quadtree is complicated for nearest neighbour search (NNS). NNS in a quadtree has to make proximity calculations to two split planes, sort the distances and select the appropriate sequence of traversal for every traversed node. On the contrary, k-d trees can be constructed so as to be balanced and symmetrical.

The experimental results also show that the performance of BIH in highly dynamic crowd simulations is not as good as in the case of ray tracing. There are two reasons for this. First, BIH can be highly imbalanced due to uneven distribution. Second, the BIH needs to be rebuilt at each frame and the complexity of building a BIH is higher than for a k-d tree.

In our experiments, the cell size of a grid is always pre-defined, so if the scene space is changed, the grid needs to be re-defined. The k-d tree, quadtree and BIH do not pre-define those values; they can be re-calculated during run time.

Theoretically, when the leaf node in a tree structure contains only one object, we can get best performance by avoiding most of the unnecessary

aspects of collision detection. Howeer, our experimental results show that this is not necessarily true. A k-d tree and quadtree have their best performance when each leaf node contains 1% of the total number of objects, and 0.5% for BIH. The reason behind this is that there has to be a trade-off between collision detection and neighbour searching time. When the tree structure has exceeded the depth, more time is spent on neighbour searches for each object than on checking collision. On the other hand, the best performance we could achieve was when cell size was equal to the EOBB size of an object, using a grid data structure. Typically, each cell should only contain one object. Because the grid has a constant query time, when the number of objects in each cell is small, the need for assessment of collision possibilities will decrease immediately.

Finally, it should be noted that in the presented results, optimizations such as crowd-based occlusion culling are not applied as the aim of our research is to compare the performance between each data structure for large-scaled crowd. However, the optimizations can be easily implemented into the simulation in future.

6.3 Performance Measures

In this section, we are going to implement the proposed method using several reality models, and measure the evacuation time for each case. Because no real data was available, approximations had to be made about how fast people would traverse the environment. We assumed the width of a pedestrian was 50cm and the maximum walking speed was 2 m/s while slow walking speed was 1 m/s, respectively.

6.3.1 A Large Hall(E3)

In the first case, E3 (discussed in Chapter 3) is used, as shown in Fig. 6.24.. Both width and length are 25m, and the exit width is 4m. One obstacle is located in the hall. A different view of the scene is shown in Fig. 6.34 before 200 pedestrians were added to the hall. The evacuation time is measured and the results are discussed in the following sections.

Based on our simulation model, the evacuation time has been estimated

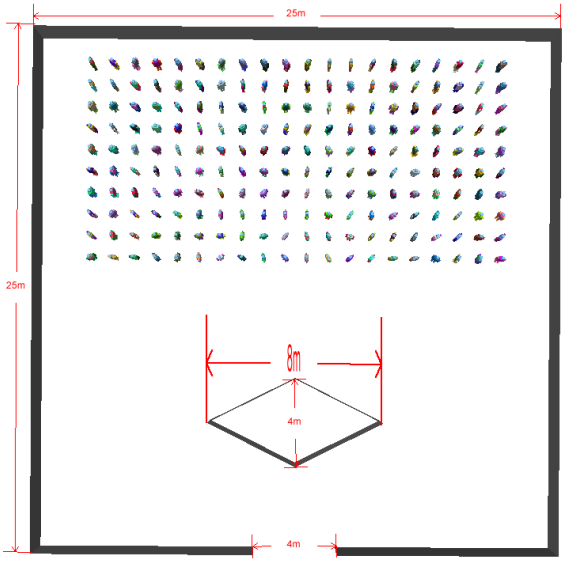

Figure 6.33: Screen-shot of a hall with an obstacle.

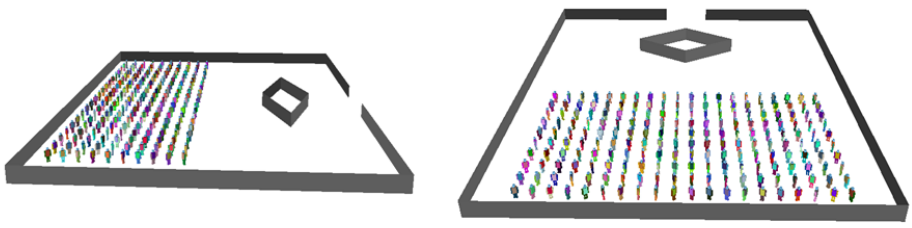

Figure 6.34: Different screenshot views of the hall with an obstacle.

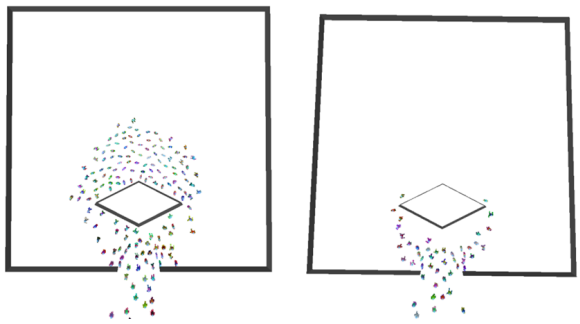

Figure 6.35: Screenshot of different stages of simulation for evacuating the hall. The left image shows how most of the pedestrians are blocked by the obstacle. The right image shows pedestrians leaving the hall.

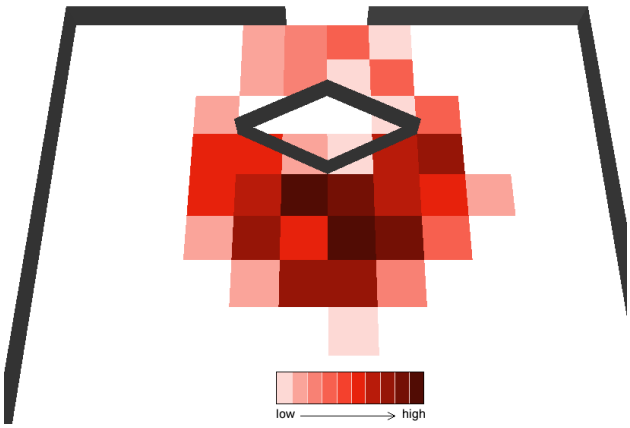

Figure 6.36: Density map after the simulation runs for approximately 150 seconds. the direction changes frequently around the obstacle, which is also the most congested area.

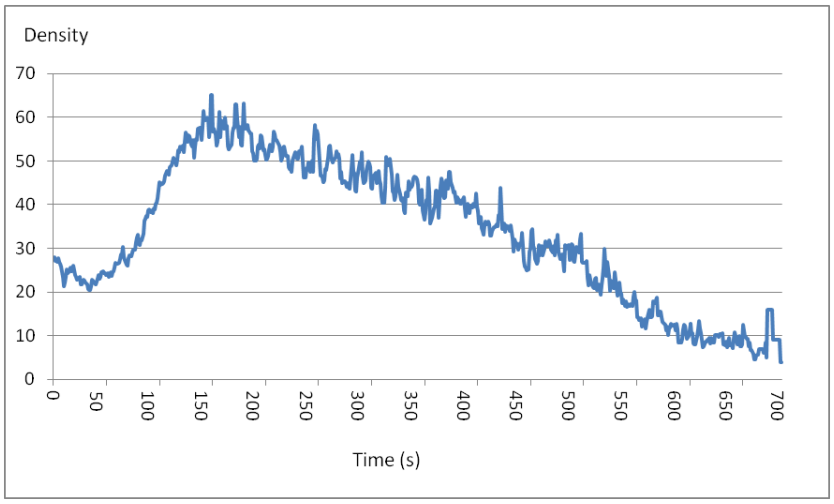

Figure 6.37: Density changes. Density reaches a peak after running the simulation for 150s, indicating the path was congested with crowd.

as 730s to evacuate the entire hall. The screenshot of simulation for each stage is shown in Fig. 6.35. The density map after running the simulation for 150s is shown in Fig. 6.36. It can be observed that high densities and clutter are located near the obstacle. We also measured the density based on the simulation time. Fig. 6.37 30 illustrates the density value for different stages of the simulation. The peak value for density is achieved after the simulation runs for approximately 150s. Most of the facts and observations are the same ,as shown in Fig. 6.36, and the crowd is the most congested during that time.

6.3.2 A Lecture Theatre (E4)

A more complicated model is designed and the top-down view of the environment is shown Fig. 6.38. The model has a corridor in the middle of the scene. Each side of the corridor has five large rooms and four obstacles are present in the corridor. The room size is 17m $\times$ 25m, and the door of room is 4m wide. The corridor width is 17m, and the exit size is 6m. 200 pedestrians are assigned in each room and the evacuation time is measured.

The simulation took 15 minutes to evacuate all pedestrians from this

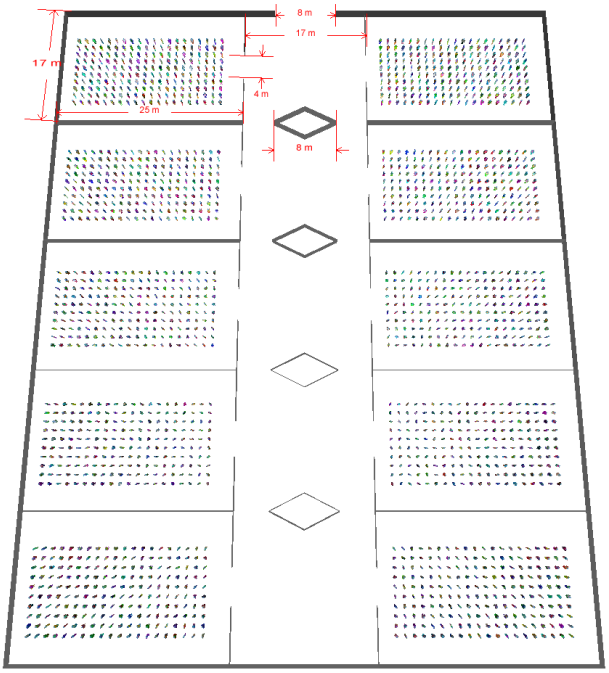

Figure 6.38: Top-down view of the large lecture theatre with obstacles in corridor.

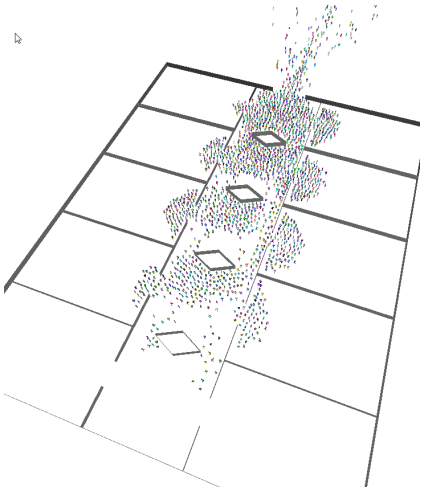

Figure 6.39: Start of the convergence in the corridor as pedestrians moved closer to the exit of the lecture room.

exhibition centre. Different stages of the simulation are shown in Fig. 6.39, Fig. 6.40, and Fig. 6.41. The results show the motion planning algorithm still work perfectly in this complicated environment with large numbers of pedestrians. The pedestrians can find the free path and leave the build smoothly (Fig. 6.41). And based on the results, the bottleneck of this environment can be predicted based on the simulation results. Fig. 6.39 shows that the pedestrians start converging in front of the exit of the build at the beginning of the simulation. Fig.6.40 shows that even though most of the lecture rooms are empty, large numbers of pedestrians are still converged in the corridor. The most congested area is the same throughout the simulation. In the next step, we modified the building structure to analyse the exact bottleneck.

As mentioned previously, to pin point the exact bottleneck involved, we removed all obstacles from the corridor and ran the simulation again (Fig. 6.42). The evacuation time remained at 15 minutes, the same as the previous case when there were obstacles in the corridor.This indicates that the obstacles in the corridor are not the bottlenecks. It can also be said that the corridor could in no way be a bottleneck with or without obstacles, as the time for evacuation remained the same, without any major locked-in situa-

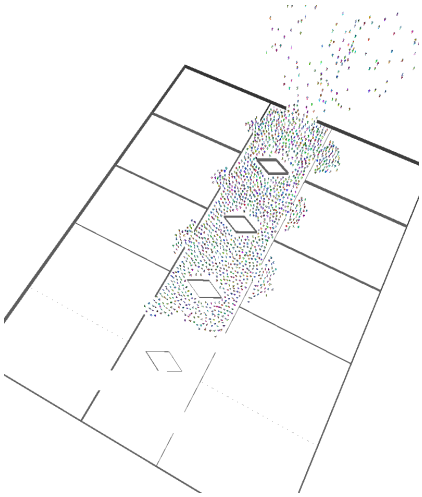

Figure 6.40: Most of the rooms are empty and the pedestrians converge in the corridor.

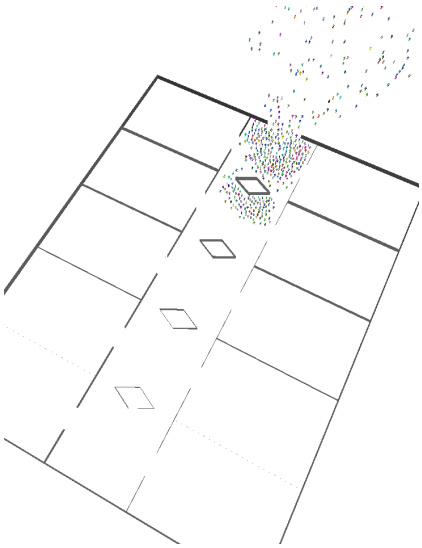

Figure 6.41: Most of the pedestrians having evacuated the building. The congestion can still be seen at the front of the exit

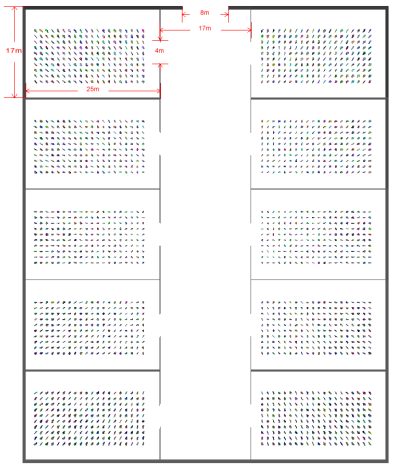

Figure 6.42: Top-down view of the large lecture theatre without obstacles.

tions or congestions. Hence we concluded that the exit could be the most likely bottleneck of this build.

In order to test if the exit is the problem, we enlarged the door size to 17m (Fig. 6.43). It is noticed that the evacuation time is reduced drastically: it took just 7 minutes to evacuate when the hall has an exit 17m wide. It is also observed that there are no major convergences near the exit. Now that the size of the exit problem has been cleared, the obstacles could be viewed as a bottleneck (Fig.6.45). The screenshots of different stages of the simulation are depicted Fig. 6.44, and Fig. 6.45. To visualize the results, density maps are also used, where colouring denotes the density level for different cases. Not surprisingly, the theatre with a larger exit has a well-distributed density (Fig. 6.46) and the density near the exit and the top obstacle is slightly higher. Fig.6.47 and Fig.6.48 show that the theatre has a similar density distribution with or without obstacles, and the highest density can be observed near the exit.

Finally, we checked different crowd sizes in the three different situations(narrow exit with obstacles in the corridor, narrow exit without obstacles in the corridor and wide exit). The results are shown as a graph in Fig.6.49. It shows that the theatre with a large exit always has the best performance than the others. It can also be observed that there is not much

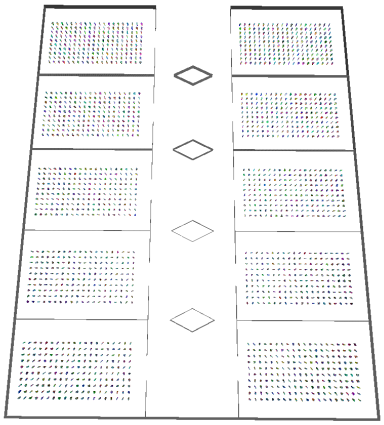

Figure 6.43: Top-down view of the large lecture theatre with a larger exit door.

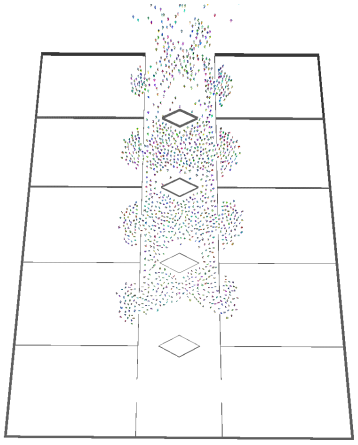

Figure 6.44: Pedestrians converged in the corridor. No congestion in front of the exit.

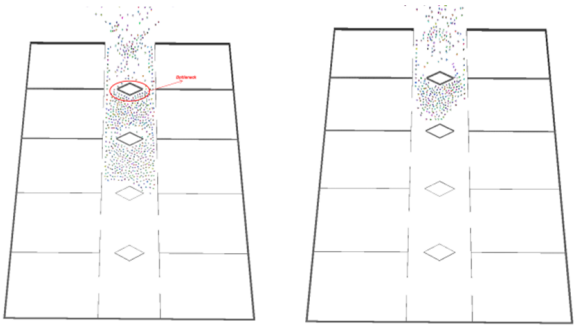

Figure 6.45: Sequence screenshots of simulation after the exit is enlarged. The congested area is now the first obstacle (red circle).

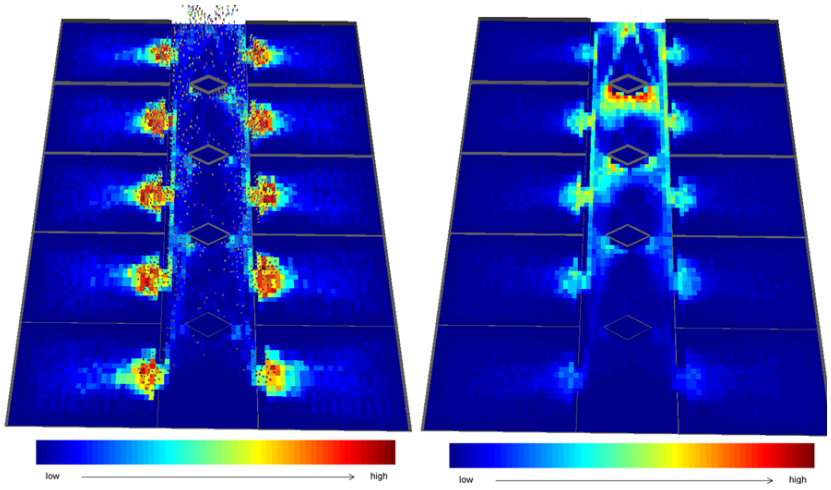

Figure 6.46: Density map of the theatre with a wide exit. Not too high a density in this situation.

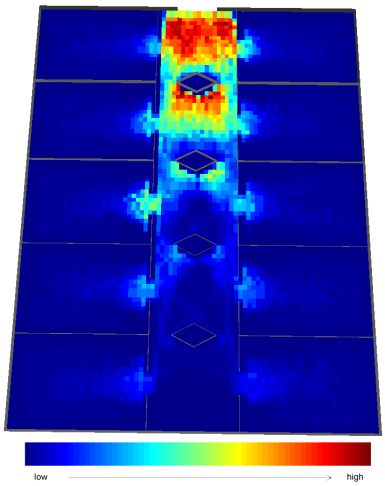

Figure 6.47: Density map of the theatre with obstacles. Congested areas are near the exit and the first obstacle.

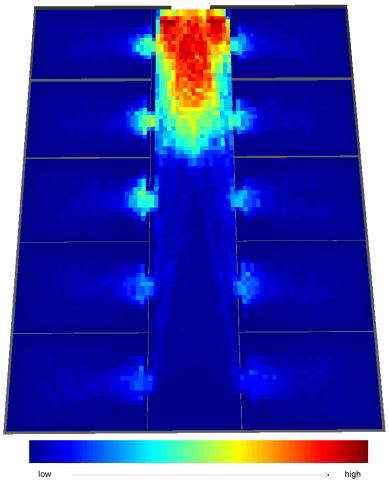

Figure 6.48: Density map of the theatre without obstacles. High density near the exit.

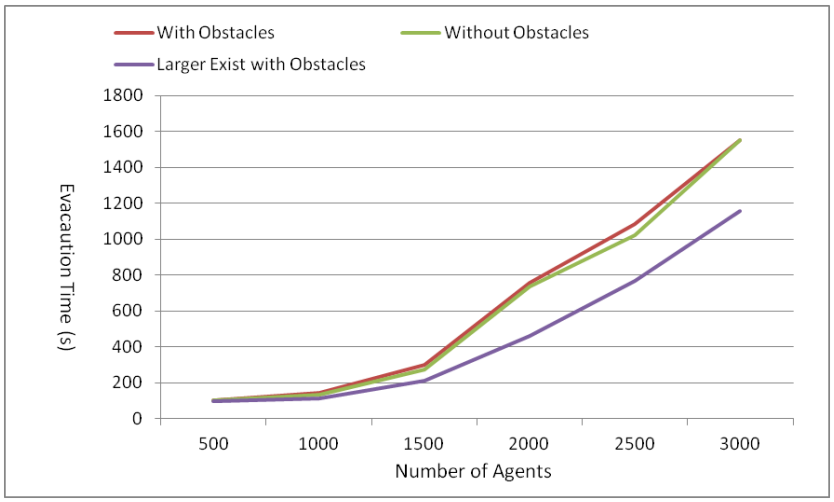

Figure 6.49: Lecture room evacuation times measured under different situations

difference in performance when the crowd size is less than 1,000.

In this section, we have used our proposed approach to predict the evacuation times. Since we do not have any real data, all parameters in our experiment are assumed. The measured time may not be reasonable. However, the experimental results show that the proposed approach is able to analyze the builds infrastructure and investigate the bottlenecks of the building.

6.4 Compared with Other Methods

To evaluate and show the effectiveness of the simulation model described in this work, a comparative analysis between different crowd simulation models is performed. The results are shown in Table 6.1 The comparison shows that our proposed model has good functionality. In contrast with other methods, only Wen Tangs method and our proposed method can simulate large numbers of pedestrians in real-time. The acceleration data structure that we have used is necessary to run a large-scaled crowd simulation in real-time, where as Wen Tang and Wee Lit Kohs methods used quadtree and partitioning of the space methods. In our work, we proved that grid has much better performance than quadtree. In our model, we were able to

	Our Method	Qiu Qins method [48]	Wee Lit Kohs method [66]	Wen Tangs method [70]
Behaviour model	Rule-based	Social-force	Rule-based	Rule-based
Model of the Crowd	macroscopic and microscopic model	macroscopic model	macroscopic and microscopic model	macroscopic model
Global Path Finding	No	topological graphs and roadmaps	A* search algorithm	A* search algorithm
Motion Planning	Three EOBBs help the agent determine the valid velocities and avoid collision	social force	vision model, sampling system by using the density	sub-goal is defined by proximity checking
Pedestrians Model	3D visual model	Particle	Particle	Particle
Environment	2D, 3D	2D	3D	3D
Large-scaled crowd simulation	Yes, 10,000 agents	No	No	Yes, 1,000 agents
Real-time	yes	yes	yes	yes
Acceleration data structure	Grids, Quadtree, k-d tree, BIH	No	Quadtree	BSP tree, Quadtree
Bounding box	EOBB	Noicle	No	No
Collision Detection	OBB-OBB	Proximity distance querycle	Proximity distance query	Proximity distance query
Obstacle Avoidance	OBB-AABB	Proximity distance query	Proximity distance query	Proximity distance query

Table 6.1: Comparison between different crowd simulation models

run real-time simulations with 10,000 pedestrians, but in Wen Tangs method simulation in real time can be done with only 1,000 pedestrians.

Our method can display both detailed and macro levels of the crowd. Wee Lit Kohs method also implemented both microscopic and macroscopic models. The difference is that Wee Lit Kohs has used particles to represent the pedestrians, where as we used actual pedestrians. Wee Lit Kohs model uses simple proximity distance checks and hence the collision detection between each pedestrian cannot be achieved at a detailed level. However in our implementation, at the detailed level the bounding box is used for pedestrians and OBB-OBB collision detection is performed, which can detect the collision possibility accurately. Other methods also used particles, but they are not effectual as they cannot investigate the behaviour of an individual pedestrian.

Each method has different motion planning algorithms. Qiu Qins model uses a social force algorithm. The motion of agents is calculated by adding several social force terms together. In high-density crowd situations, their effect is either repulsive or attractive. However, it may not produce realistic results in certain situations. Wee Lit Koh uses a vision model to define the motion. They sample the people around an agent's location, and use the distance check to find the free path. The sampling of a number of the people is increased when the density is high. This method can simulate motion more realistically than our method, but the complexity of their method is also much higher. Wen Tangs motion planning algorithm is quite simple. The agents are defined as a group instead of individuals: the agents in each group follow the leader with their sub-goal, and the sub-goal is calculated by checking the proximity distance. Their method is very easy to implement, but the drawback is that deadlock can happen when the crowd density is high, as discussed in Section 6.1.

Visualizations are also an important factor for crowd simulation, and this is the reason we used both 2D and 3D to represent our model. Similar 3D models have been used by Wen Tang and Wee Lit Koh for simulation. In Wen Tangs model, highly detailed pedestrian models are used which provide better visualization but reduce the performance of the simulation. One major drawback in our simulation model is that it does not have a global path

finding scheme. Due to constraint of time we could not implement any algorithm for global path finding. This can be implemented easily in the future and is recommended as an extension for our work.

6.5 Development Environment

All applications are developed in C++ programming language. Microsoft Visual Studio 2005(SP1) is used as the Integrated Developments Environment. OpenGL API is used for rendering 2D and 3D graphics for the simulation. Along with the C++ Standard Template Library, glut library is used for OpenGL programming. All experiments have been run on a single PC. The hardware or configuration details of the PC are: Intel Core(TM) 2 Quad CPU Q6600 (at 2.40 GHz) processor, 3 GB of RAM and GeForce 9500GT graphics card with 256 MB onboard DDR-2 memory. The operating system was Windows XP Professional 32-bit with Service Pack 3 and DirectX 9.0c.

Chapter VII

Conclusions

The aim of this thesis was first to present a new data structure and related methods for fast collision detection and obstacle avoidance in the simulation of crowd motion, then to compare four acceleration data structures in an attempt to achieve large-scale crowd simulation in real-time. In this chapter we summarize our contributions in real-time crowd simulations and discuss future work.

7.1 Summary and Conclusions

Crowd simulation application. In the first part we built a 3D crowd simulation application. The application was built based on C++ and OpenGL. Object-oriented programming was used and many flexible control functions were implemented to the application. The goal was to allow the designer to edit simulation easily and investigate the simulation visualized. The outcome of the application proved to be very positive, as shown in Chapter 6. With this crowd simulation application, different behaviour models and the acceleration data structures could be easily added. The simulation was built in a 3D model so the visualization of the simulation was more realistic. The simulation could be paused and run at any time, and the angle of the camera could be changed to any direction so we could observe different viewpoints of the crowd simulation. The zoom in and out function provided the view of the micro and macro model of simulation, as well as the flow fields and density maps. The flow field graphs showed visually convincing motions of crowds directly, and density maps were used to visualize the density distribution clearly. The simulation core was simple but generic, which made it possible to attach modules for all kind of behaviours and simulation features.

EOBB and Obstacle Avoidance. In the second section, a new data structure called Extended Oriented Bounding Box (EOBB) was designed, which encapsulated geometry and details of the motion of agents into a simple bounding-box structure. It could also detect collisions between pedestrians or pedestrians and obstacles faster and more accurately. Our experiments confirmed the assertion that the use of EOBB accelerated collision testing among many characters, an outcome that improved the efficiency demand.

Motion Planning. I n the third section, we developed efficient a collision avoidance and motion planning algorithm to help agents find free paths. In our research, agents were not expected to know everything; they did not have a priori information about the static and/or dynamic obstacles they would encounter along their path, they were not bound by crowd group decisions, and they were allowed to make individual motion plans. We assigned simple constrains to the agents' motion, then used the EOBB structure to bound agents with those motions with three possible directions, which could efficiently find the free path and resolve deadlock conditions. The primary benefit of this algorithm is that it was easy to implement and could be used effectively in several contexts, such as collision detection and obstacle avoidance. The implementation yielded satisfactory results in several scenarios where collision avoidance and directional updates were important.

Acceleration Data Structures. In the last section, four spatial partitioning data structures were implemented. The aim was to find the best acceleration data structures for crowd simulation, so they could be run real-time with large crowd sizes. We presented an overview of the four commonly used spatial subdivision methods and analysis using update time with respect to variations in crowd size, grid cell size, and the maximum number of objects in the leaf nodes of the tree structures for quadtree, k-d tree and the bounding interval hierarchy (BIH). An extensive comparative analysis of the performance of spatial subdivision structures in large-scale crowd simulations was then presented.

The simulation results showed that a grid data structure with EOBBs for character models gave the best performance when the number of crowd members was very large. Crowd simulation of 10,000 agents in real-time could only be achieved by using such spatial data structures. The grid data structure had the advantage that it did not need to be updated, and an efficient hash map implementation could provide fast look-up.

Our methods were appropriate for a variety of virtual environments. For example, a virtual shopping mall or virtual theatre environment could be easily populated with crowds that react to the target environment appropriately. This populating could be realized through the installation of specific rules into the environment. In Chapter 6, we provided the experimental results of measuring evacuation time and finding the bottleneck of a building. The outcomes showed our algorithm could help to predict actual crowd movement before a buildings construction is started.

As well as those applications, our approach is quite useful for the development of a large-scale crowd simulation in real-time. The experimental results showed that our application could run in real-time with 10,000 agents.

7.2 Improvements and Further Work

Further research and expansion are suggested in this section.

Rendering

In our application, we rendered the whole scene at each frame, no matter how much data was contributed to the given frame. This can have a high computational cost when the scene is complicated.

In future work, level of detail (LOD) and culling method could be implemented into the application. Since we simulated crowds at the micro and macro level, the LOD technique could reduce many unnecessary computations. Another way to reduce the computational cost would be to use a culling method so that only geometry that will contribute to a given frame would need to be rendered. Many culling methods have

been proposed which can significantly reduce the amount of rendering data. For example, the view frustum culling [61] and occlusion culling [42] have been proposed. Culling methods can even combine with the spatial partitioning data structure to give better performance.

Reality Model

We have not used any textures in rendering our scenes, and the pedestrian model was made up of simple cubes. All those aspects could be improved in the future so the simulation could look more realistic. For example, we could texture all obstacles and walls, and use a mesh model to represent a variety of pedestrians.

Path Finding

In our experiment, pedestrians were primarily concerned with navigating in their immediate neighbourhood while avoiding other agents and obstacles. Our experimental results showed that agents in the models can perceive their local surroundings and adjust their behaviours accordingly, and that motion planning methods are sufficient for local collision avoidance. However, agents could get stuck when exclusively relying on local models in complex environments. In the future, path finding algorithms such as A* algorithm or Dijkstra's algorithm, could be implemented.

Acceleration Data Structures

Future work in this area should be directed towards combining collision avoidance with path/motion planning algorithms incorporating various types of behaviour models. Effective mechanisms for improving the performance of hash mapping for the grid structure could also be explored.

Behaviour Model The proposed method could be further extended to implement several directional updates (instead of just three) for motion planning and perspective modelling. Another important area of future wok is to incorporate specific aspects of crowd behaviour into the motion update algorithm. For example, agents should have different

moving speed for genders, and when accidents happen, agents should have some specific behaviour in evacuation simulation.

Bibliography

[1] *Real-time collision detection.* Amsterdam, 2005.

[2] E. Angel, *Interactive computer graphics : a top-down approach using OpenGL.* Addison Wesley, 2006, chapter 10.

[3] L. Bentley, J, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, p. 509517, 1975.

[4] Bortoffs, "Path planning for uavs," *2000 American Control Conference*, p. 364368, 2000.

[5] U. Branislav, H. C. Pablo, de, and T. Daniel, "crowd brush interactive authoring of real-time crowd scenes," *SIGGRAPH 2005 Courses*, 2005.

[6] M. Breivik and T. Fossen, "Principles of guidance-based path following in 2d and 3d," *44th IEEE International Conference on Decision and Control*, p. 627634, 2005.

[7] K. Cain, Y. Chrysanthou, and F. Silberman, "A case study of a virtual audience in a reconstruction of an ancient roman odeon in aphrodisias," *SIGGRAPH 2005 Courses*, 2005.

[8] W. Carsten and K. Alexander, "Instant ray tracing: The bounding interval hierarchy," *Proceedings of the 17th Eurographics conference on Rendering Techniques*, pp. 139–149, 2006.

[9] R. Craig, W, "Flocks, herds and schools: A distributed behavioral model," *The 14th annual conference on Computer graphics and interactive techniques*, pp. 25–34, 1987.

[10] T. Daniel and M. Soraia, Raupp, *Crowd Simulation.* Springer, 2007.

[11] B. David, C and H. Jessica, K, "Simulation level of detail for multiagent control," *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pp. 199–206, 2002.

[12] L. David and R. Martin, "Level of detail for 3d computer graphics," *Elsevier Science Inc*, 2002.

[13] F. Ding, P. Jiao, X. Bian, and H. Wang, "Auv local path planning based on virtual potential field," *IEEE International Conference on Mechatronics and Automation*, p. 17111716, 2006.

[14] P. M. Dirk Helbing, "Social force model for pedestrian dynamics," *Review E*, pp. 4282–4286, 1995.

[15] B. Dutra, T, B. Cavalcante-Neto, J, and A. Vidal, C, "A hybrid model for simulating crowd with varied behaviors in real-time," *Proceedings of the 13th Symposium on Virtual Reality*, pp. 232–241, 2011.

[16] M. Eitz and L. Gu, "Hierarchical spatial hashing for real-time collision detection," *IEEE International Conference on Shape Modeling and Applications*, pp. 61 – 70, 2007.

[17] M. Erik and R. Isaac, "Impostors and pseudo-instancing for gpu crowd rendering," *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pp. 49–55, 2006.

[18] S. F-Frisken and N. Perry, R, "Simple and efficient traversal methods for quadtrees and octrees," *Journal of Graphics Tools*, 2002.

[19] W. Fule, Z. Changle, and H. de, Garis, "A simple robot paths planning based on quadtree," *IEEE International Conference on Progress in Informatics and Computing*, pp. 1–4, 2010.

[20] Z. Gabriel, "Minimal hierarchical collision detection," *Proceedings of the ACM symposium on Virtual reality software and technology*, 2002.

[21] V. Havran, "Heuristic ray shooting algorithms," Ph.D. dissertation, Czech Technical University in Pragueol, 2001.

[22] V. Havran and J. Bittner, "On improving kd-trees for ray shooting," *WSCG 2002 Conference*, p. 2002, 209216.

[23] P. Hoogendoorn, S and H. L. Bovy, P, "Generic gas-kinetic traffic systems modeling with applications to vehicular traffic flow," *Transportation Research Part B*, p. 2001, 317336.

[24] B. Hou and B. Wang, "Crowd psychology simulation incorporating psychometrics and intervention of relationship spaces," *Proceedings of the International Conference on Advances in System Simulation*, pp. 155–160, 2009.

[25] S. Hu and L. Yu, "Optimization of collision detection algorithm based on obb," *Measuring Technology and Mechatronics Automation n*, pp. 853–855, 2010.

[26] K. James, T, H. Martin, B. M. Joseph, S, S.-r. Henry, and Z. Karel, "Efficient collision detection using bounding volume hierarchies of k-dops," *IEEE Transactions on Visualization and Computer Graphics*, p. 2136, 1998.

[27] D. Jan, J. Joran, and T. Harry, "A multi-agent cellular automata model of pedestrian movement," *Pedestrian and Evacuation Dynamics*, pp. 173–181, 2002.

[28] S. Javed, N. Ehsan, G. Javed, Salman, Z. Sarwar, Shaikh, and E. Mirza, "Computational simulation of a macroscopic traffic model for highways in pakistan," *IEEE International Conference on Management of Innovation and Technology*, pp. 1182–1187, 2010.

[29] Y. Jihyun and D. Crane, C, "Development of a sensor knowledge store using a quadtree algorithm to support autonomous vehicle path plan-

ning," *International Conference on Control Automation and Systems*, pp. 1508–1512, 2010.

[30] R. Kadowaki, K. Motomura, S. Ohkura, and K. Aizawa, "Graphs representing quadtree structures using eight edges," *2010 4th International Symposium on Communications, Control and Signal Processing*, pp. 1–5, 2010.

[31] J. Kalojanov and P. Slusallek, "A parallel algo-rithm for construction of uniform grids," *Proceedings of the Conference on High Performance Graphics*, p. 2328, 2009.

[32] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *IEEE International Conference on Robotics and Automation. Proceedings*, p. 9098, 1986.

[33] Y. Kitamura, T. Tanaka, F. Kishino, and M. Yachida, "3-dpath planning in a dynamic environment using an octree and an artificial potential field," *IEEE International Conference on Intelligent Robots and Systems*, p. 474481, 1995.

[34] A. Lagae and P. Dutr, E, "Compact, fast and robust grids for ray tracing," *Computer Graphics Forum (Proceedings of the 19th Eurographics Symposium on Rendering)*, p. 2008, 12351244.

[35] D. Lee, T and C. Wong, K, "Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees," *Acta Informatica*, p. 2329, 1977.

[36] J. Levinthal, C, "Molecular model-building by computer," *Scientific American*, p. 4252, 1966.

[37] J. Li-jun, C. Jin-chang, and Z. Wei-jie, "A crowd evacuation simulation model based on 2.5-dimension cellular automaton," *Virtual Environments, Human-Computer Interfaces and Measurements Systems*, pp. 90–95, 2009.

[38] D. MacDonald, J and S. Booth, K, "Heuristics for ray tracing using space subdivision," *Graphics Interface Proceedings*, p. 152163, 1989.

[39] B. Mirtich, "Efficient algorithms for two-phase collision detection," *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, p. 1998, 203223.

[40] R. Mukundan, *Advanced Methods in Computer Graphics.* Springer, 2012.

[41] J. Nadira, A. Naida, and A. Elma, *Dijkstra's shortest path algorithm serial and parallel execution performance analysis.* Proceedings of the 35th International Convention, 2012.

[42] G. Naga, K, S. Avneesh, Y. Eui, and M. Dinesh, "Interactive visibility culling in complex environments with occlusion-switches," *Proceedings of the 2003 symposium on Interactive 3D graphics*, p. 2003, 103 - 112.

[43] B. Nam, "A comparative study of spatial indexing techniques for multidimensional scientific datasets," *Proceedings. 16th International Conference on Scientific and Statistical Database Management*, 2005.

[44] P. Nuria and M. Ali, "Comparison of crowd simulations for building evacuation and an alternative approach," *Proceedings of Building Simulation*, pp. 1514–1521, 2007.

[45] C. O'Sullivan, J. Cassell, and H. Vilhjlmsson, "Levels of detail for crowds and groups," *Computer Graphics Forum*, pp. 733–741, 2002.

[46] M. Overmars, H, "Point location in fat subdivisions," *Information Processing Letters*, p. 261265, 1992.

[47] Potuzak and Tomas, "Comparison of road traffic network division based on microscopic and macroscopic simulation," *Computer Modelling and Simulation*, pp. 409–414, 2011.

[48] Q. Qiu and W. Junhu, "An agent-based approach for crowd dynamics simulation," *IEEE International Conference on Intelligent Computing and Intelligent Systems*, pp. 78– 82, 2010.

[49] Z. Qu, J. Wang, and C. Platsted, "A new analytical solution to mobile robot trajectory generation in the presence of moving obstacles," *IEEE Trans. Robot*, p. 978993, 2004.

[50] H. Samet, "Neighbor finding techniques for images represented by quadtrees," *Computer Graphics and Image Processing*, pp. 35–57, 1982.

[51] S. Sarmady, F. Haron, and Z. Talib, A, "Simulating crowd movements using fine grid cellular automata," *Proceedings of the 12th International Conference on Computer Modelling and Simulation*, pp. 428–433, 2010.

[52] M. Schevtsov, A. Soupikov, and A. Kapustin, "Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes," *Computer Graphics Forum*, p. 395404, 2007.

[53] Sharma and Sharad, "Crowd simulation in emergency aircraft evacuation using virtual reality," *Proceedings of the 16th International Conference on Computer Games*, pp. 12–17, 2009.

[54] F. Shen, W. Hui, and L. Hong, "A mesoscopic model for bicycle flow," *30th Chinese Control Conference*, pp. 5574–5577, 2011.

[55] G. Stephen, J, C. Jatin, C. Sean, D. Pradeep, L. Ming, and D. Manocha, "Pledestrians: a least-effort approach to crowd simulation," *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 119–128, 2010.

[56] L. Steven, M, *Planning Algorithms*. Cambridge University Press, 2006.

[57] M. Sung, S. Chenney, and Gleicher, "Detecting collisions in graph-driven motion synthesis," Computer Sciences Technical Report CS-2005-1529, university of Wisconsin.

[58] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross, "Optimized spatial hashing for collision detection of deformable objects," *Vision, Modeling, Visualization*, p. 47 54, 2003.

[59] D. Thalmann, L. Kermel, W. Opdyke, and S. Regelous, "Crowd and group animation," *Proceeding SIGGRAPH '05 ACM SIGGRAPH Courses*, 2005.

[60] G. Turk, "Interactive collision detection for molecular graphics," Master's thesis, The University of North Carolina, 1989.

[61] A. Ulf and M. Tomas, "Optimized view frustum culling algorithm for bounding boxes," *Journal of Graphics Tools*, pp. 9–22, 2000.

[62] B. Ulicny and D. Thalmann, "Crowd simulation for interactive virtual environment and vrtraning systems," *Proceedings of Eurographics workshop on Animation and Simulation*, p. 163170, 2001.

[63] T. Vaisagh, V and M. Lees, "An information-based perception model for agent-based crowd and egress simulation," *Proceedings of the International Conference on Cyberworlds*, pp. 38–45, 2011.

[64] H. Vic, "Opengl tutorial: Quaternion c amera class," 2011, http://www.nehe.gamedev.ne.

[65] I. Wald, S. Boulos, and P. Shirley, "Ray tracing deformable scenes using dynamic bounding volume hierarchies," *ACM Transactions on Graphics*, 2007.

[66] K. Wee, Lit, L. Lin, and Z. Suiping, "Modelling and simulation of pedestrian behaviours," *22nd Workshop on Principles of Advanced and Distributed Simulation*, pp. 43–50, 2008.

[67] S. Wei and T. Demetri, "Autonomous pedestrians," *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 19–28, 2005.

[68] E. Weisstein, "Cellular automation," *MathWorld–A Wolfram Web Resource*, 2011, http://mathworld.wolfram.com/CellularAutomaton.html.

[69] C. Wen, S. Hui, Z. Shulong, and Z. Baoshan, "Application of an improved a* algorithm in route planning," *WRI Global Congress on Intelligent Systems, 2009*, pp. 253–257, 2009.

[70] T. Wen, W. Tao, Ruan, and P. Sanket, "Real-time crowd movement on large scale terrains," *Theory and Practice of Computer Graphics*, pp. 146– 153, 2003.

[71] B. Wenfeng, H. Lina, and S. Yunfeng, *An improved a* algorithm in path planning.* International Conference onComputer, Mechatronics, Control and Electronic Engineering, 2010.

[72] W. Xiao-rong, W. Meng, and L. Chun-gui, "Research on collision detection algorithm based on aabb," *Natural Computation*, pp. 422–424, 2009.

[73] M. Xiong and M. Lees, "A rule-based motion planning for crowd simulation," *Proceedings of the International Conference on Cyberworlds*, p. 2009, 88-95.

[74] F. Zhigang, J. Jianxun, and X. Jie, "Efficient collision detection using a dual k-dop-sphere bounding volume hierarchy," *IFITA*, p. 2010, 185 189.